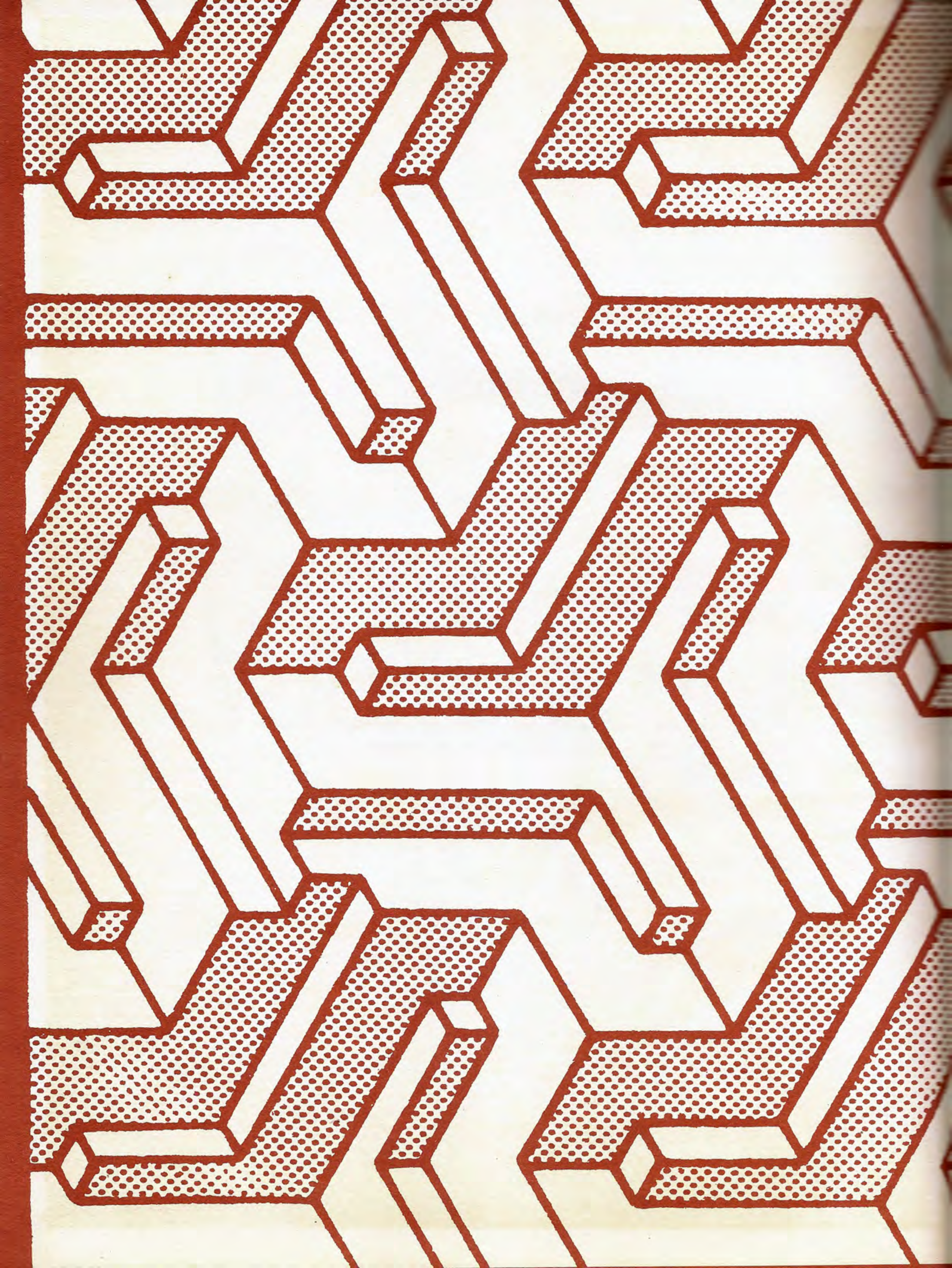
# GRAN ENCICLOPEDIA INFORMATICA

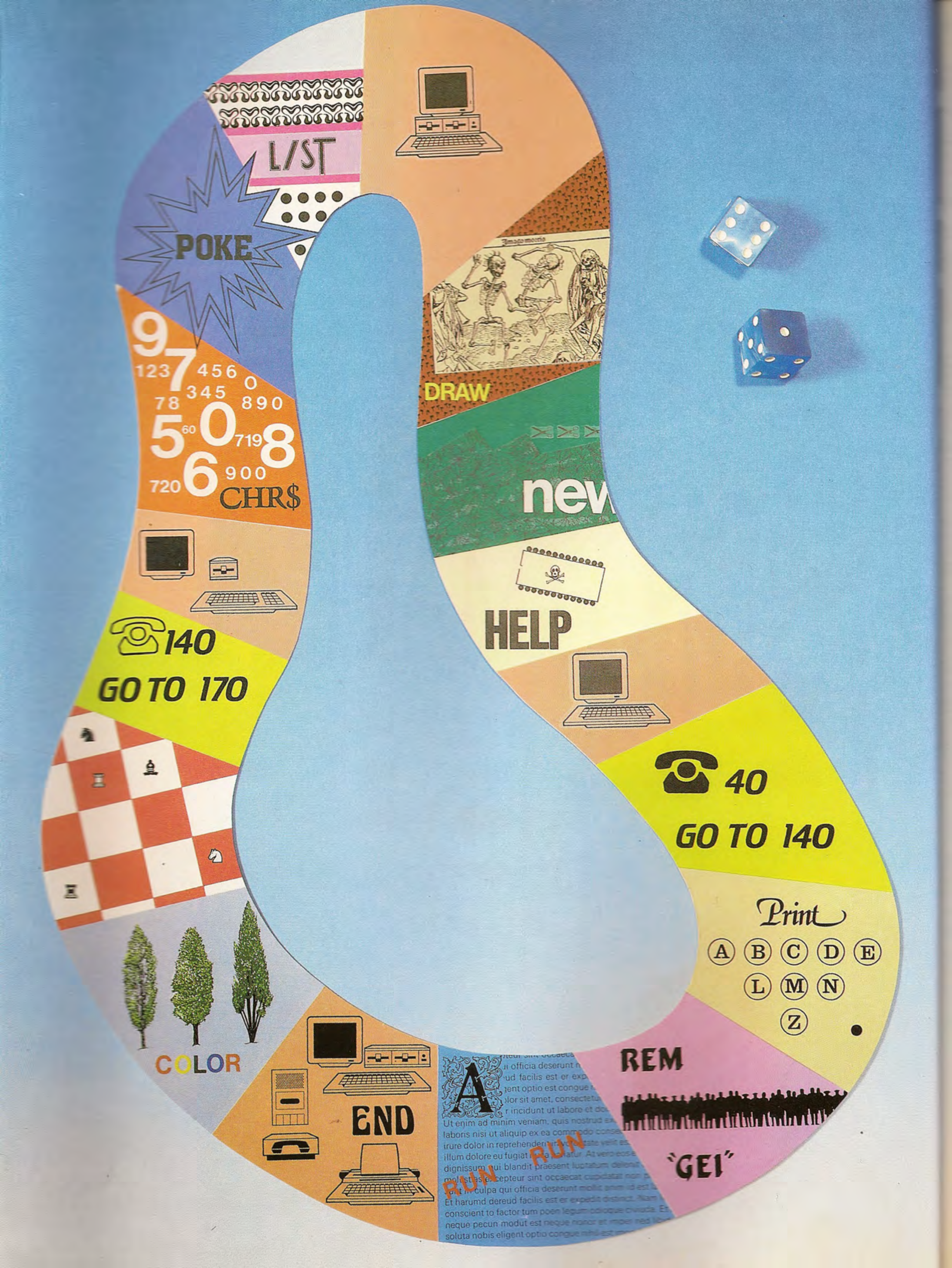
## CURSO DE BASIC/ 3

EDICIONES NUEVA LENTE



# GRAN ENCICLOPEDIA INFORMATICA

EDICIONES NUEVA LENTE



L/ST
POKE
9 7 5 0 8 6
123 456 0
345
78 890
60 719
720 900
CHR$
DRAW
new
HELP
140
GO TO 170
40
GO TO 140
Print
A B C D E
L M N
Z
COLOR
END
RUN
RUN
REM
"GET"

# SUMARIO

*Una publicación:*

**Ediciones Nueva Lente, S. A.**

*Director editor:* MIGUEL J. GOÑI

*Director de producción:* SANTOS ROBLES.

*Director de la obra:* FRANCISCO LARA.

*Colaboradores:* PL/3 - MANUEL MUÑOZ - ANGEL MARTINEZ - MIGUEL DE ROSENDO - DAVID SANTAOLALLA - SANTIAGO RUIZ - LUIS COCA - MIGUEL ANGEL VILA - MIGUEL ANGEL SANCHEZ VICENTE ROBLES.

*Diseño:* BRAVO/LOFISH.

*Maquetación:* JUAN JOSE DIAZ SANCHEZ.

*Ilustración:* JOSE OCHOA.

*Fotografía:* (Equipo Gálata) ALBINO LOPEZ y EDUARDO AGUDELO.

*Ediciones Nueva Lente, S. A.:*
Dirección y Administración:
Benito Castro, 12. 28028 Madrid. Tel.: 245 45 98.

*Números atrasados y suscripciones:*
Ediciones Ingelek, S. A.
Plaza de la Rep. Ecuador, 2 - 1.°. 28016 Madrid.
Tel.: 250 58 20.

*Plan general de la obra:*
18 tomos monográficos de aparición quincenal.

*Distribución en España:*
COEDIS, S. A. Valencia, 245. Tel.: 215 70 97.
08007 Barcelona.

*Delegación en Madrid:*
Serrano, 165. Tel.: 411 11 48.

*Distribución en Argentina:*
*Capital:* AYERBE
*Interior:* DGP

*Distribución en Chile:* Alfa Ltda.

*Distribución en México:*
INTERMEX, S. A.
Lucio Blanco, 435
México D.F.

*Distribución en Uruguay:*
Ledian, S. A.

*Edita para Chile:*
PYESA
Doctor Barros Borgoño, 123
Santiago de Chile

*Importador exclusivo Cono Sur:*
CADE, SRL. Pasaje Sud América, 1532.
Tel.: 21 24 64. Buenos Aires - 1.290. Argentina.



*Fotomecánica:* Ochoa, S. A.
Miguel Yuste, 32. 28037 Madrid.

*Impresión:* Gráficas Reunidas, S. A.
Avda. de Aragón, 56. 28027 Madrid.

ISBN de la obra: 84-7534-184-5.
ISBN del tomo 10: 84-7534-207-8

*Printed in Spain*
Depósito legal: M. 27.605-1986



Precio de venta al público en Canarias, Ceuta y Melilla: 940 ptas.
Enero 1987

# El ordenador útil

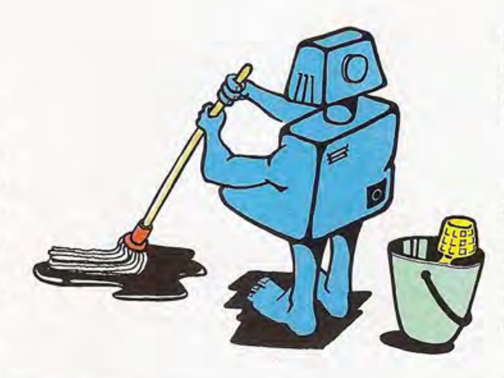
## Programando aplicaciones

Tras dos tomos dedicados fundamentalmente a la explicación de conceptos, en este tercero nos ocuparemos básicamente de la práctica del BASIC.

En anteriores capítulos prácticos se presentaron ejemplos en los que se hacía uso de los comandos BASIC para fines fundamentalmente lúdicos. En éste utilizaremos la potencia del BASIC para resolver algunos problemas cotidianos.

Un ordenador es una máquina de calcular. Sus posibilidades, sin embargo, superan a las de cualquier calculadora. Con un ordenador se pueden realizar tareas mucho más complejas y con mayor facilidad. Y es para eso, precisamente, para lo que está pensado el ordenador.

A lo largo del presente capítulo se construirá una base de datos. Esta no será ni mucho menos tan potente como una base de datos comercial; sin embargo, la podremos diseñar a nuestra medida y, desde luego, resultará bastante más económica.



*El ordenador útil ha de colaborar con el usuario en sus labores cotidianas.*

### Una base de datos

En principio, una base de datos consiste en un conjunto organizado y más o menos voluminoso de datos. Estos datos mantienen ciertas relaciones entre sí. Un ejemplo de base datos lo constituye el propio listín telefónico. En él coexisten dos tipos de datos que se relacionan mutuamente: el nombre del abonado y el número de su teléfono. En general, una base de datos puede disponer de más tipos de datos, denominados normalmente «campos». Cada grupo de datos diferentes recibe el nombre de «ficha» o «registro». De esta forma, un listín telefónico puede constar, por ejemplo, de 100 registros de dos campos.

El método más adecuado para guardar esta información pasa por el uso de un «array» o matriz. Se puede definir un array de dos dimensiones, 100×2, asegurándose así el espacio suficiente para los datos. Al tratarse de datos numéricos y alfanuméricos no se puede emplear una matriz numérica. Lo más conveniente, pues, es crear la matriz como un array de caracteres, transformando los datos numéricos en alfanuméricos (función STR$). Si más adelante fuera necesario realizar cálculos con los números almacenados, éstos serían reconvertidos previamente con la función VAL.

En una base de datos son necesarias al menos tres operaciones: creación, actualización y visualización de los datos. La primera es obvia, sin crearla no existiría tal base de datos. Una vez creada será necesario actualizarla de tiempo en tiempo; esto es, modificar datos o introducir otros nuevos. Y todo ello no sirve de nada si no es posible consultar esos datos, visualizándolos en pantalla o volcándolos en la impresora. Acometeremos pues estos tres procesos viendo las distintas posibilidades que ofrece el BASIC para su programación.

### La creación

A la hora de crear la base de datos es cuando se ha de dimensionar la matriz correspondiente. En este punto los ordenadores difieren unos de otros, dependiendo de la forma en la que tratan los datos de tipo cadena de caracteres. Los ordenadores Sinclair y Atari almacenan las cadenas en forma de arrays, conteniendo cada elemento un sólo carácter. De esta forma, si A$ contiene la cadena "JAUME" el valor de A$(3) será "U".

Esto no ocurre en los restantes equipos. Lo normal es almacenar una cadena entera como elemento del array. Si en el BASIC de Sinclair y Atari la orden DIM A$(5) reserva espacio para cinco carateres, en los demás permitirá almacenar cinco cadenas de caracteres de longitud ilimitada.

Atendiendo a estas diferencias, el dimensionamiento de la matriz puede presentar dos aspectos. Para el caso de una matriz de 100×2, en la que cada cadena ha de tener un máximo de 20 caracteres, sus formulaciones serían las siguientes. En los equipos Atari y Sinclair:

```
10 DIM A$(100,2,20)
```

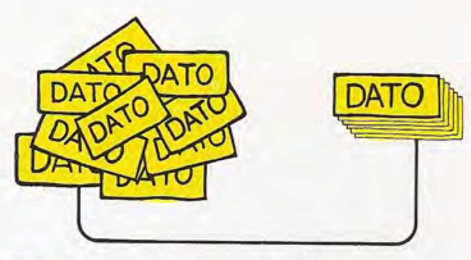
Mientras que en los restantes dialectos BASIC:

```
10 DIM A$(100,2)
```

A lo largo del presente capítulo nos acogeremos a la segunda formulación, más genérica. No obstante, si su ordenador pertenece a los nombrados, no se olvide de realizar las modificaciones indicadas. Como tamaño general de la base de datos elegiremos 100 registros de 10 campos cada uno. Con ese presupuesto, la línea de dimensionado se reduce a:

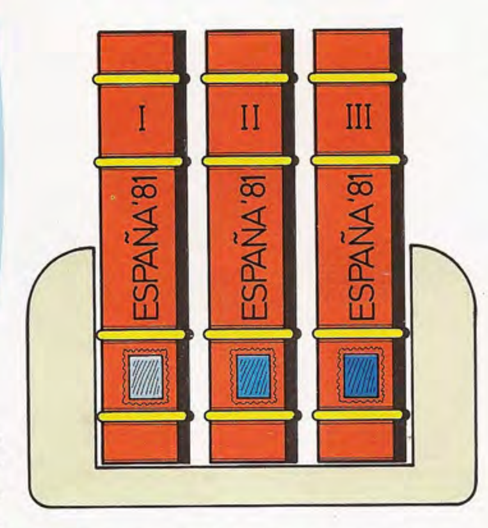
```
10 DIM A$(100,10)
```

En ella se indica que el primer subíndice corresponde a los registros y el segundo a los campos. Una vez dimensionada la matriz, hay que proceder a «rellenarla» con datos. Estos se han de introducir por medio del teclado, de ahí que sea conveniente utilizar el comando INPUT. Para recorrer los campos de todos los registros se hace uso de un doble bucle.




*El manejo y organización de los datos es una de las tareas en las que el ordenador se manifiesta como un consumado especialista.*

```
1000 FOR REG=1 TO 100
1010 FOR CAM=1 TO 10
1020 INPUT B$
1030 LET A$(REG,CAM)=B$
1040 NEXT CAM
1050 NEXT REG
■
```




*Nuestra base de datos servirá para guardar información de forma organizada y accesible.*

Este irá pidiendo los datos uno a uno hasta completar toda la tabla. Tal y como está codificada esta rutina puede llegar a aburrir el ver 100×10=1000 veces el signo "?" mientras introducimos los datos. Lo más adecuado será visualizar un indicador del elemento que se está introduciendo. Así, al mismo tiempo tendremos constancia directa del dato que introducimos y donde lo hacemos. Para lograrlo basta con introducir una línea intermedia en el programa:

```
1015 PRINT "REGISTRO: ";REG;"CAMPO: ";CAM
```

De esta forma, la ejecución de esta primera rutina daría el siguiente resultado en pantalla:

```
RUN<CR>
REGISTRO: 1 CAMPO: 1
?MANUEL
REGISTRO: 1 CAMPO: 2
?2340700
..................
REGISTRO: 100 CAMPO: 10
? FIN
■
```




*Para seleccionar la operación a realizar en la base de datos se hace uso de un menú de opciones ejecutables.*

En la mayoría de los casos no se hará uso de todos los registros y campos. Por lo tanto, no será imprescindible rellenarlos todos. Para evitar la molestia de pasar por todos los campos de cada registro es preciso introducir nuevas líneas. En ellas se ha de inspeccionar el dato introducido y, si se trata de una orden de salto de registro o de fin de tarea, realizar dichas acciones.

Para indicar el final de la entrada de datos hemos de teclear la orden al efecto. Esta consistirá en un conjunto de caracteres plenamente diferenciados de un dato ordinario. Por ejemplo, puede emplearse un carácter especial, poco usual, que sea identificado como indicativo de un comando. Cualquier carácter especial (#, $, %, *, £, @, etc.) puede servir al efecto. En nuestro caso, utilizaremos el carácter "@"; si usted no dispone de este carácter o no resulta de su agrado, puede utilizar otro, siempre y cuando lo tenga en cuenta en las siguientes líneas de programa.

En definitiva, podemos definir el comando de "fin de entrada de datos" como "@F" (F de fin), y el de salto de registro como "@ R" (R de registro).

Ahora hay que lograr que el programa contemple estas posibilidades. El salto de registro implica pasar al primer campo del siguiente registro, lo que significa hacer CAM=1 y REG=REG+1. Sin embargo, como existe una línea NEXT CAM y otra NEXT REG, habrá que poner los valores anteriores (los comandos NEXT incrementan la variable a la que hacen referencia). Por consiguiente, los valores adecuados son CAM=10 y REG tal como estaba. Esto habrá que realizarlo cuando se haya tecleado el comando apropiado (@ R). La línea en cuestión adoptará el aspecto siguiente:

```
IF B$="@ R" THEN LET CAM=10
```

Esta ha de colocarse después de la lectura de B$ y antes de trasladar dicho valor al array (línea 1030).

No interesa que el «comando» se vea reflejado en la base de datos como un dato más; por ello, en tal caso, se ha de saltar la línea 1030. Asi pues la correcta programación del citado «comando» coincidirá con:

```
1025 IF B$="@ R" THEN LET CAM=10:GOTO 1040
```

Por lo que respecta al final de la entrada de datos, el método varía muy poco. La línea correspondiente difiere tan sólo en los nuevos valores de CAM y REG. Para que el bucle se dé por terminado, estos valores han de coincidir con los valores límite de cada bucle FOR: 10 y 100, respectivamente:

```
1027 IF B$="@ F" THEN LET CAM=10: LET REG=100:GO
     TO 1040
```

Con esta línea queda «casi» completa la rutina de creación de la base de datos. Esta será utilizada a modo de subrutina, por lo que será necesario completarla con una línea en la que figure un comando RETURN (retorno al punto de origen del salto a subrutina). El listado completo es el que aparece a continuación.

```
  10 DIM A$(100,10)
1000 FOR REG=1 TO 100
```

```
1010 FOR CAM=1 TO 10
1015 PRINT "REGISTRO: ";REG;"CAMPO: ";CAM
1020 INPUT B$
1025 IF B$="@R" THEN LET CAM=10: GOTO 1040
1027 IF B$="@F" THEN LET CAM=10:LET REG=100:GOTO
     1040
1030 LET A$(REG,CAM)=B$
1040 NEXT CAM
1050 NEXT REG
1100 RETURN
```

## Edición

Después de crear la base de datos será necesario, con cierta frecuencia, alterar alguno de los datos. Es posible que se hayan cometido errores al introducir los datos por vez primera, o simplemente que los datos deban ser actualizados.

Para cambiar el contenido de un campo específico hay que localizar el registro al que pertenece y, por supuesto, el campo implicado dentro de este registro. El método más directo para cambiar el contenido será asignar el nuevo valor, recogido en un INPUT, al elemento correspondiente. Ello se consigue por medio de la siguiente rutina.

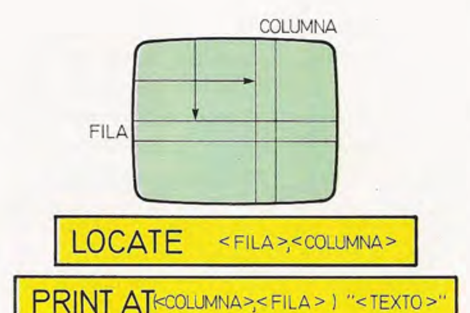
```
2100 INPUT B$
2110 LET A$(REG,CAM)=B$
```

En todo caso, es obvio que antes de ello habrá que depositar los valores adecuados en REG y CAM. Estos dos datos pueden también captarse a través de sendas instrucciones INPUT. Por lo demás, también es conveniente mostrar el contenido original del campo y su identificación. Todo ello queda contemplando en la siguientes líneas de programa:

```
2010 INPUT "REGISTRO";REG
2020 INPUT "CAMPO";CAM
2030 PRINT "REGISTRO";REG;"CAMPO";CAM
2040 PRINT "CONTENIDO: "A$(REG,CAM)
```

Ahora resultará fácil la edición de los datos contenidos en la base. Se elige un registro y un campo, se ve su contenido y se introduce el nuevo valor.

La ejecución de la rutina que nos ocupa producirá un resultado similar al que sigue:




*El comando LOCATE constituye una alternativa para aquellos ordenadores que no disponen del comando PRINT AT.*

```
RUN<CR>
REGISTRO? 12<CR>
CAMPO? 3<CR>
REGISTRO 12 CAMPO 3
CONTENIDO: RUIIZ
? RUIZ
■
```

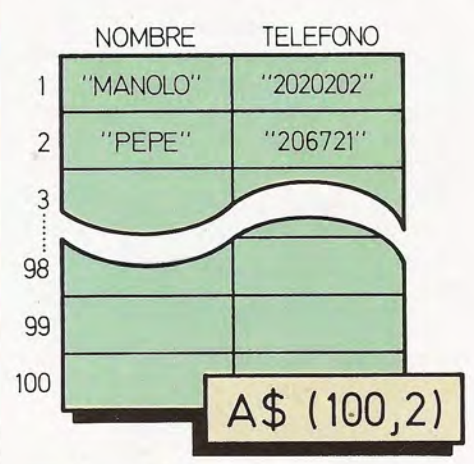
Desde luego, aún cabe realizar algunas mejoras en la rutina de edición. Es conveniente contar con la posibilidad de dejar inalterado el contenido de un campo, en el caso de que sea el correcto. También resultaría interesante la posibilidad de editar más de un campo de un tirón.

Para solventar el primer problema es suficiente con introducir una condición; por ejemplo: si no se introduce ningún valor, el contenido del campo en cuestión no debe ser alterado. Dicho efecto se consigue con la línea siguiente:

```
2104 IF B$="" THEN GOTO 2120
```

Esta nueva instrucción hace que cuando no se introduzca un dato se salte a la línea 2110, en la que se asigna el nuevo valor. El salto incondicional (GOTO) se efectúa a una línea que todavía no ha sido escrita; ésta debe ser la que permita editar varios campos en una misma operación.

Una vez introducidos los datos de registro y campo que determinan el dato a editar (líneas 2010 y 2020), se pueden seguir editando los campos que aparezcan a continuación. Para ello basta con modificar el valor de REG y CAM. Esta acción debe realizarse inmediatamente




*Nuestra base de datos, confeccionada a título de ejemplo, emplea un «array» de cadenas de caracteres para almacenar organizadamente los distintos datos.*

después de la edición del dato correspondiente:

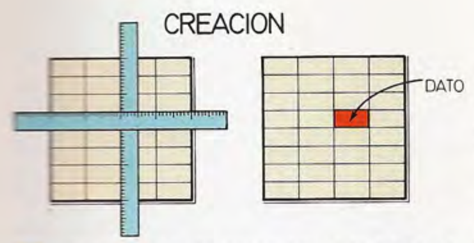
```
2120 LET CAM=CAM+1
2130 IF CAM=11 THEN LET CAM=1:LET REG=REG+1
```

### LOCATE

Sitúa el cursor en el lugar de la pantalla indicado en su argumento.

*Formato:* (n.º de línea) LOCATE <fila>, <columna>

*Ejemplos:*
```
10 LOCATE 10,8 : PRINT "HOLA"
50 LOCATE F,C : PRINT DATO
```



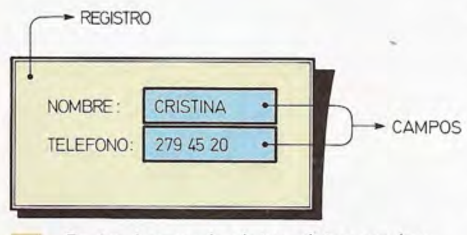

*En la etapa de creación de la base de datos, dimensionaremos la matriz de almacenamiento y procederemos a la introducción de los primeros datos.*

```
2140 IF REG<101 THEN GOTO 2030
2150 PRINT "NO HAY MAS DATOS"
```

La línea 2120 es la encargada de incrementar el contador de campos. En el caso de agotar los campos de un registro, se ha de pasar al siguiente registro. Ello se ordena en la línea 2130. Mientras no se acaben los registros (mientras REG sea menor que 101) se regresará a la zona de edición, mostrando el campo y pidiendo un nuevo dato (GOTO 2030). En el caso de llegar al final de la base de datos aparecerá en la pantalla el mensaje "NO HAY MAS DATOS".

Al igual que en la etapa de creación de la base, en la edición puede no ser necesario recorrer todos los datos. En este punto, introduciremos dos «comandos» que harán más cómoda esta zona del programa. El primero de ellos servirá para abandonar la edición, el otro permitirá saltar al siguiente registro.

Para mantener un paralelismo con los «comandos» empleados en la zona de creación, utilizaremos los mismos caracteres. Así pues, para dar por terminada la edición se tecleará "@F". La línea detectora de este comando se formula como sigue:




*En las bases de datos, éstos suelen organizarse en registros: a su vez, cada registro acoge a los datos guardándolos en los respectivos campos.*

```
2106 IF B$="@F" THEN RETURN
```

El motivo de incluir el RETURN en el argumento de THEN obedece a que, como en el caso de la creación, se trata de una subrutina.

Al tropezar con este comando, el programa volverá a la zona principal de cuyo estudio nos ocuparemos más adelante. El salto al registro siguiente se obtendrá con los caracteres "@R". En este caso, la línea correspondiente es idéntica a la utilizada en la rutina de creación.

```
2108 IF B$="@R" THEN LET CAM =1:LET REG=REG
     +11:GOTO 2030
```

Tal y como se observa, el salto (GOTO) se realiza al punto en el que se permite la edición del dato elegido.

La rutina al completo adopta el siguiente aspecto:

```
2010 INPUT "REGISTRO";REG
2020 INPUT "CAMPO";CAM
2030 PRINT "REGISTRO";REG;"CAMPO";CAM
2040 PRINT "CONTENIDO: ";A$(REG,CAM)
2100 INPUT B$
2104 IF B$="" THEN GOTO 2120
```

## Diagramas de flujo

La fase previa a la codificación de un programa se concreta en el análisis y descomposición del problema a resolver. Ello empieza con el estudio del algoritmo o método de solución, y más tarde, mediante la descomposición del algoritmo en una serie de acciones básicas reflejadas en un ordinograma.

Los diagramas de flujo, también llamados flujogramas u ordinogramas, son gráficos que permiten la representación simbólica del algoritmo. Con ello se consigue ordenar un problema en una sucesión de pasos que permitirán desglosarlo. Por lo tanto, estos flujogramas facilitan la posterior codificación e introducción en la máquina del algoritmo para resolver el problema.

Cuando un algoritmo ha sido traducido a ordinograma, es más fácil codificar el programa utilizando un lenguaje informático.

El orden lógico para confeccionar un programa destinado al ordenador se concreta en los siguientes puntos:

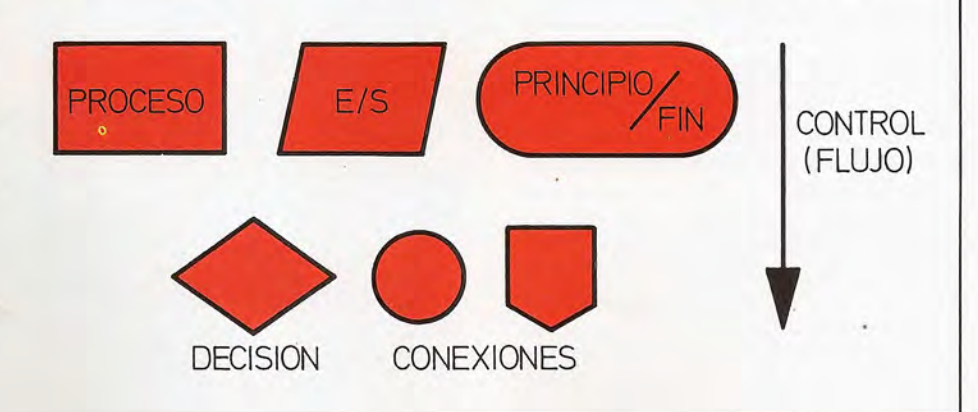
1. Estudio del algoritmo.
2. Traducción del algoritmo a ordinograma.
3. Confección del programa en el lenguaje apropiado.

Muchos programadores piensan que no es necesario pasar por la fase de construir el ordinograma. Sin embargo, la experiencia dicta que, en el 90% de los casos, tal omisión sólo contribuye a dificultar la escritura del programa. Hasta llegar incluso al extremo de imposibilitar su eficaz puesta a punto.

Los ordinogramas se componen de dos zonas fundamentales. Por un lado las órdenes o acciones que debe ejecutar el ordenador; éstas se enmarcan dentro de una serie de figuras geométricas. Por otro, la secuencia de control o de procesado de órdenes, que se representa por medio de líneas que enlazan a las diversas acciones.

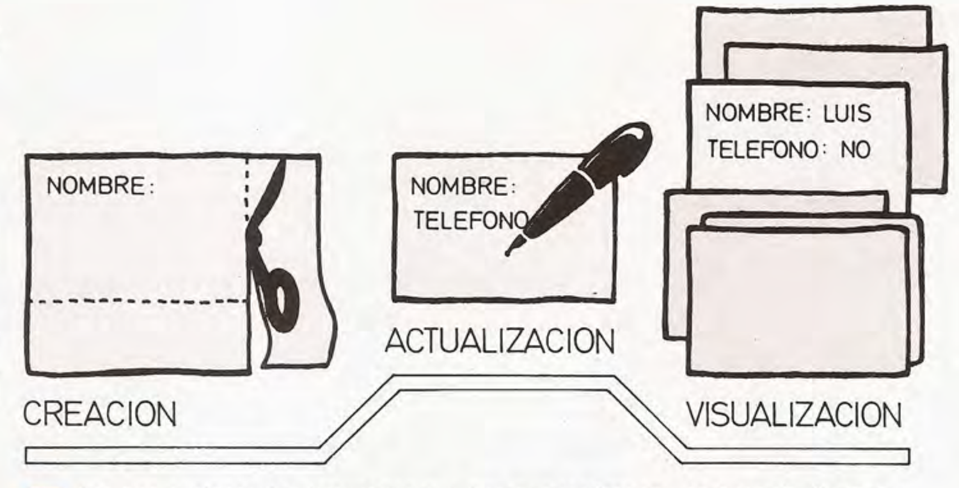
Cada tipo de orden o acción se enmarca dentro de una determinada figura geométrica. Las principales son las que ilustra la figura adjunta.

Es importante observar que el ordinograma resulta de fácil lectura y constituye una documentación del programa eficaz e inteligible. En él pueden utilizarse frases del lenguaje corriente, abreviaturas, e incluso instrucciones próximas a los lenguajes de programación.



```
2106 IF B$="@F" THEN RETURN
2108 IF B$="@R" THEN LET CAM=1: LET REG=REG+1:GO
     TO 2030
2110 LET A$(REG,CAM)=B$
2120 LET CAM=CAM+1
2130 IF CAM=11 THEN LET CAM=1:LET REG=REG+1
2140 IF REG<101 THEN GOTO 2030
2150 PRINT "NO HAY MAS DATOS"
2160 FOR I=1 TO 1000:NEXT I
2170 RETURN
```

En ella aparecen dos nuevas líneas, que introducen un retardo para que sea posible leer con comodidad el mensaje antes de regresar al programa principal.




*Las tres acciones fundamentales que hay que realizar en una base de datos se concretan en su creación, actualización del contenido y visualización del mismo.*

## Visualización

A estas alturas se han confeccionado ya las rutinas capaces de crear y editar base de datos; queda por crear una tercera zona que permita visualizar los datos. Este es, precisamente, el cometido del apartado en el que nos encontramos: estudiar la rutina de visualización.

En principio, esta rutina no tiene mayor complicación que la de dar un aspecto agradable a la presentación de los datos que, por supuesto, deben ser presentados agrupadamente, por registros. Una vez elegido el registro, éste se mostrará al completo.

Para que la presentación sea satisfactoria, han de visulizarse, junto con el contenido de los campos, el número del registro y campo de cada dato. Los distintos campos se recorren por medio de un bucle de tipo FOR/NEXT. Veamos una primera aproximación a las instrucciones de la nueva rutina.

```
3100 CLS
3110 PRINT "REGISTRO NUMERO ";REG
3120 PRINT
3130 FOR CAM=1 TO 10
3140 PRINT "CAMPO ";CAM;"=",A$(REG,CAM)
3150 NEXT CAM
```

Con esta rutina, los datos serán visualizados después de borrar la pantalla. Cada registro que se visualice ocupará toda la pantalla. Primero, como ya se ha indicado, se borra la pantalla, y tras ello se procede a mostrar el número correspondiente al registro, en la línea 3110. La siguiente línea sirve para dejar un espacio entre esa cabecera y el grupo de datos. El contenido de cada campo se visualizará en una línea de pantalla, precedido por el indicativo del campo en cuestión.

El número del registro a visualizar debe encontrarse almacenado en la variable REG. Sin embargo, esta variable no ha sido inicializada en el cuerpo de la rutina que nos ocupa. Para determinar dicho valor, se ejecutará un bucle de búsqueda cuya filosofía es la siguiente: se proporciona un dato y el ordenador busca el primer registro que contenga ese dato. Una vez localizado el número del registro afectado, se pasa a la visualización del mismo. Hay que tener en cuenta que junto con el dato a buscar debe indicarse el campo en el que se ha de realizar la búsqueda.

Veamos cómo es posible programar tal proceso:

```
3010 INPUT "BUSCO EN EL CAMPO NUM.";CAM
3020 INPUT "QUE DATO";D$
3030 FOR REG=1 TO 100
3040 IF $$(REG,CAM)=D$ THEN GOTO 3100
3050 NEXT REG
3060 PRINT "NO ENCUENTRO ESE DATO"
3070 INPUT "BUSCO OTRO DATO (S/N)";B$
3080 IF B$="S" THEN GOTO 3010
3090 RETURN
```

Las referidas líneas de programa tienen encomendada la búsqueda del dato elegido. Ello se realiza dentro del bucle FOR/NEXT delimitado por las líneas 3030 a 3050. Dicho bucle recorre todos los registros, comparando el campo mencionado con el dato propuesto. Cuando se halla un campo cuyo contenido coincide con el elegido, se procede a visualizar el registro implicado (GOTO 3100). En el caso de no encontrar el dato propuesto, concluirá el bucle mostrando el correspondiente mensaje (línea 3060). De ocurrir tal circunstancia, el programa preguntará al usuario si desea realizar otra búsqueda, para regresar, en consecuencia, a la línea 3010 o al programa principal.




*De nada sirve una base de datos si ésta no permite una adecuada presentación del contenido de los registros.*

Al concluir las líneas de programa que gestionan la visualización, hay que introducir algunas instrucciones adicionales que garanticen la continuación del programa. Siguiendo con la misma filosofía, lo lógico sería que se pasara al siguiente registro. Si se desea buscar




*La zona de visualización de la base de datos confeccionada incluye una rutina para la búsqueda de los registros.*

otro registro se empleará el comando "@R", mientras que si se opta por finalizar la inspección se tecleará "@f".

Estas nuevas posibilidades se añaden al final de la rutina de visualización. Las adecuadas líneas de programa presentan una gran similitud con las que ejecutan una función análoga en las dos rutinas precedentes:

```
3200 PRINT "ENTER..........REGISTRO SIGUIENTE"
3210 PRINT "@R..........NUEVO REGISTRO"
3220 PRINT "@F..........FINALIZACION"
3230 INPUT "OPCION";B$
3240 IF B$="" THEN REG=REG+1:GOTO 3100
3240 IF B$="@R" THEN GOTO 3010
3240 IF B$="@F" THEN RETURN
3250 GOTO 3100
```

Las tres primeras instrucciones muestran en la pantalla los comandos a utilizar y definen su función. De esta forma, el operador siempre tendrá presentes las instrucciones que puede utilizar. La línea 3230 es la encargada de captar la orden que introduzca el usuario.

## Aritmética binaria

Sin lugar a dudas, el sistema de numeración más profusamente utilizado es el decimal. Este es el habitual para el ser humano. Sin embargo, sus características lo hacen muy poco idóneo para el uso interno del ordenador. Los ordenadores utilizan el sistema de numeración binario, basado en los dígitos 1 y 0.

Para acercarse a las interioridades del ordenador resulta imprescindible conocer el sistema binario. El primer paso debe ser, forzosamente, aprender a contar en binario.

Las diferencias respecto a la representación decimal quedan patentes en la tabla adjunta. De los diez primeros números (las cifras decimales del 0 al 9), tan sólo coinciden las dos primeras. El número decimal 2 se convierte en 10 en binario, el 3 en 11, el 4 en 100, y así hasta el 9, cuya formulación binaria es 1001.

El siguiente número, esta vez ya con dos cifras, se genera sumando una unidad al anterior. Así pues, para poder seguir la cuenta es preciso conocer los fundamentos de la suma binaria.

Al igual que la suma decimal, la binaria también se realiza cifra a cifra, respetando sus posiciones respectivas; o lo que es lo mismo, operando entre sí los dígitos del mismo peso.

Por ejemplo, para sumar en decimal 17 y 35 se agrupan las unidades (7 y 5) y se suman dichos dígitos. Del resultado de esa primera operación (7+5=12) se toma el dígito de la derecha (2). El dígito sobrante (denominado de acarreo) se suma con los de las decenas (1+1+3=5). En definitiva, el resultado final es 52: cinco decenas y dos unidades.

El mismo método es el que se aplica en la suma binaria. Como en binario sólo existen dos dígitos, resulta muy fácil aprender la «tabla de sumar». Dicha tabla es la que se relaciona a continuación:

0+0=0
0+1=1
1+0=1
1+1=10

Las tres primeras expresiones coinciden con sus equivalentes en el sistema decimal. Cosa que no ocurre con la cuarta: al sumar 1 y 1 se obtiene un resultado 0 con acarreo de 1 (1+1=0... y me llevo una).

| + | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 3 | | | | | | | | |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |

DECIMAL

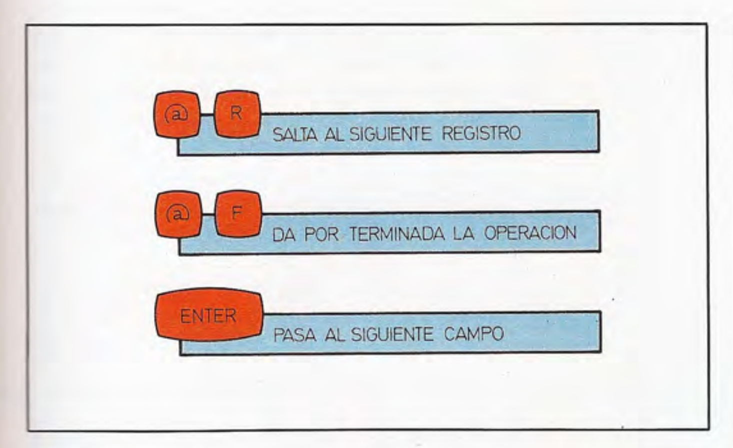
| + | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 10 |

BINARIO

| x | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

BINARIO

*Los ordenadores operan internamente en el sistema binario.*




*Dentro de nuestro ejemplo se procede al diseño de algunos pseudo-comandos destinados a facilitar el manejo de la base de datos.*



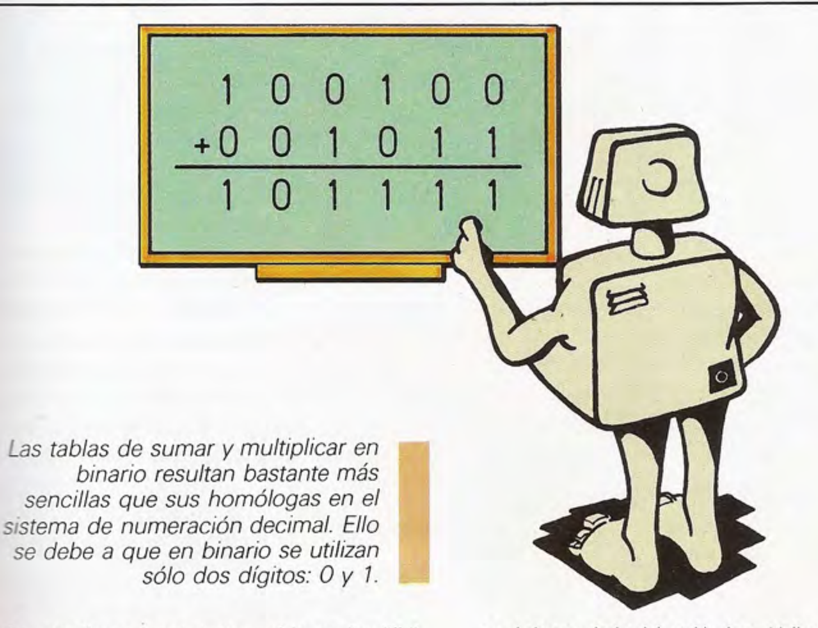

*La segunda etapa se concreta en la modificación de los datos residentes en la base, caso de que fuera preciso, y en la introducción de nuevos datos.*

A su vez, los comandos IF que siguen desvían la ejecución dependiendo de la orden recogida. Si el usuario no teclea ninguno de los comandos propuestos, se llega a la línea 3250 cuyo cometido es repetir la visualización.

## Ahora todo junto

Las tres rutinas comentadas gestionan las tres acciones enunciadas al comienzo del presente capítulo. Sin embargo, éstas han sido concebidas como subrutinas de un programa principal. Lo que queda por programar es precisamente el programa principal.

Dado que las rutinas anteriores son independientes entre sí, el cuerpo prin-




*Las tablas de sumar y multiplicar en binario resultan bastante más sencillas que sus homólogas en el sistema de numeración decimal. Ello se debe a que en binario se utilizan sólo dos dígitos: 0 y 1.*

Con estos ligeros conocimientos no debe resultar difícil sumar números de cualquier cantidad de dígitos. Por ejemplo, los números binarios 10 y 11: primero los dígitos de menor peso 0+1=1, luego los siguientes 1+1=10. Ordenando el resultado queda 101. Un valor totalmente lógico, ya que 10 es dos y 11 es tres, su suma ha de ser 101 (el cinco decimal).

La multiplicación binaria tampoco encierra misterio alguno. El método vuelve a ser semejante al utilizado en el sistema decimal. La tabla de multiplicar binaria es la siguiente:

0×0=0
0×1=0
1×0=0
1×1=1

A partir de esta tabla se puede ya realizar cualquier multiplicación binaria. Como ejemplo se muestra una sencilla multiplicación de números de dos cifras

```
  10
 ×11
 ---
  10
 10-
 ---
 110
```

Al igual que en el caso de la suma, tampoco ahora fallan las matemáticas: tres por dos es igual a seis en cualquier sistema de numeración.

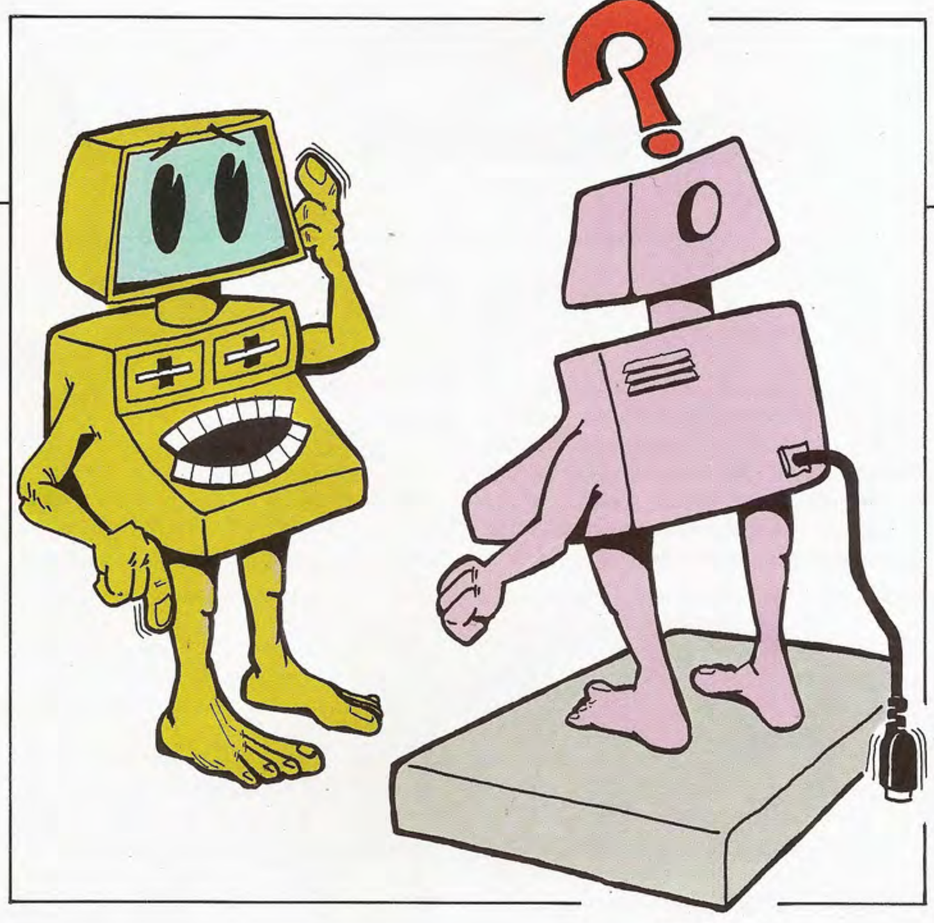
Conociendo la suma y la multiplicación binarias, la división y la resta son fácilmente deducibles. A partir de ahí trabajar en binario es tan sencillo como hacerlo en decimal.

| DECIMAL | BINARIO |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 10 |
| 3 | 11 |
| 4 | 100 |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |
| 8 | 1000 |
| 9 | 1001 |

cipal del programa accederá a cada una de ellas de forma independiente. En definitiva, el cuerpo principal del programa debe proponer al usuario la elección entre uno de los tres procesos: creación, edición o visualización. De ello se encargará un «menú» visualizado en la pantalla.

La confección de menús es una tarea ya descrita en otro punto de la obra. De ahí que no profundicemos al respecto en este capítulo. Por el contrario, sí vamos a estudiar un nuevo comando que en determinados ordenadores realiza la misma función que el conocido PRINT AT.

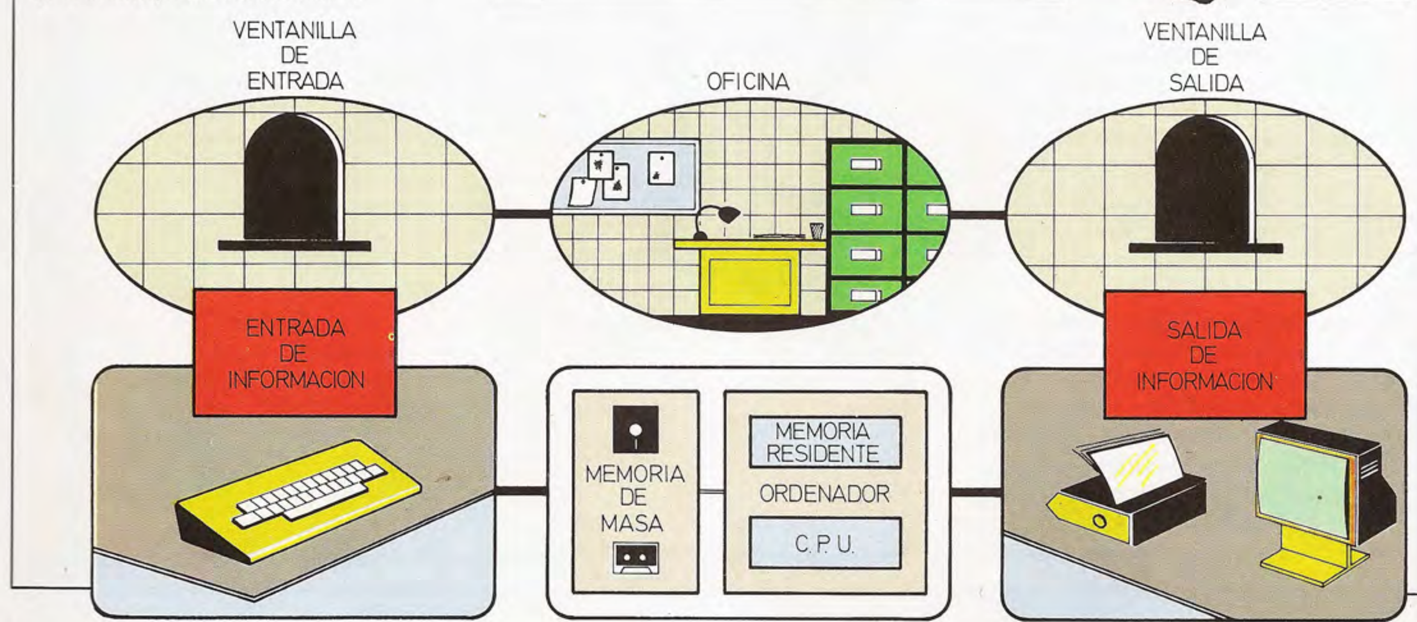
Se trata del comando LOCATE, cuya especialidad es situar el cursor en un determinado punto de la pantalla. Tras



## La actividad del ordenador

¿Para qué sirve un ordenador? Este es uno de los primeros interrogantes que surgen al tomar un primer contacto con el mundo de la informática. La respuesta no deja de resultar problemática por su difícil concreción. El motivo radica en que un ordenador puede hacer, o por lo menos ayudar a hacer, casi todo; siempre que cuente con el programa adecuado y con el apoyo de los periféricos idóneos.

A la hora de adquirir una idea real de lo que cabe esperar de un ordenador, es muy conveniente empezar conociendo cómo trabaja. Para ello, resulta ilustrativo comparar la actividad de la máquina con el trabajo cotidiano en la oficina de facturación de una empresa. La oficina está situada en una habitación provista de dos ventanillas, una de recepción o entrada y otra de salida. El empleado necesita tener a mano el conjunto de normas que detallan su trabajo, así como los datos necesarios para realizar cada tarea. Para ello dispone de un tablero, donde coloca el albarán que contiene los datos de entrada. Este albarán incluye el nombre del cliente con el que se efectúa la operación, así como el tipo de artículos y la cantidad de los mismos.



ejecutarlo, el siguiente comando PRINT empezará a escribir a partir de dicho punto. En los dialectos BASIC que incluyen el comando LOCATE en lugar del PRINT AT, la instrucción PRINT AT 2,4 "HOLA" habrá de desdoblarse en dos: la de posicionamiento y la de impresión. La primera se resuelve con LOCATE 4,2 y la segunda con PRINT "HOLA". Obsérvese que los parámetros de LOCATE están intercambiados con respecto a los de la partícula AT. La razón de ello la

| TABLA DE CONVERSION | | |
|---|---|---|
| | Posicionamiento en pantalla | |
| **Ordenador** | **PRINT AT (X, Y)<arg>** | **LOCATE X,Y** |
| **AMSTRAD** | — | LOCATE X,Y |
| **APPLE II (APPLESOFT)** | -(2) | — |
| **APRICOT (M-BASIC)** | — | — |
| **ATARI** | — | POSITION X, Y |
| **CBM 64** | — | — |
| **DRAGON** | PRINT <n>,<arg> (1) | — |
| **EQUIPOS MSX** | — | LOCATE X, Y |
| **HP-150** | — | — |
| **IMB PC** | — | LOCATE Y, X |
| **MPF** | —(2) | — |
| **NCR DM-V (MS-BASIC)** | — | — |
| **NEW BRAIN** | — | — |
| **ORIC** | PRINT X, Y; <arg> | — |
| **SHARP MZ-700 (MZ-BASIC)** | — | — |
| **SINCLAIR QL** | — | CURSOR X, Y |
| **SPECTRAVIDEO** | — | LOCATE X, Y |
| **ZX-SPECTRUM** | PRINT AT Y, X; <arg> | — |

(1) En el Dragon, el número <n> especifica una posición de la pantalla.
(2) Utilizan HTAB y VTAB para indicar las coordenadas horizontal y vertical respctivamente.

encontramos en la distinta formulación de este nuevo comando:

(Número de línea) LOCATE <fila>, <columna>

Como ilustración práctica del uso del comando, y siguiendo con el ejemplo desarrollado en el presente capítulo, se incluyen a continuación las primeras líneas del menú.

```
200 REM MENU
210 LOCATE 5,10
220 PRINT "MENU"
230 LOCATE 10,5
240 PRINT "1..........ENTRADA DE DATOS"
250 LOCATE 12,5
260 PRINT "2..........EDICION DE DATOS"
270 LOCATE 14,5
280 PRINT "3..........VISUALIZACION DE DATOS"
```

Las referidas instrucciones llevarán a la pantalla las opciones de que dispone el usuario.

Recuerde que si su ordenador no incluye el comando LOCATE, puede realizar la misma función con PRINT AT. Por supuesto, también puede adecuar el menú a su pantalla variando las coordenadas de LOCATE (o de PRINT AT).

Sobre el tablero también hay un folleto con la tarifa de precios vigente y con información sobre los descuentos a aplicar a cada cliente. Al lado de estos datos se encuentra la lista de normas a seguir para realizar el trabajo. Algo semejante a un *programa* cuyas instrucciones detallan, paso a paso, las operaciones a poner en práctica. Como herramienta de cálculo, el administrativo dispone de una pequeña calculadora. Dado el volumen de la información puesta en juego, es posible que el espacio disponible en el tablero sea insuficiente. Por esta razón el operario dispone de un archivo localizado en una habitación contigua. En este almacén residen los ficheros que guardan la información que no es posible mantener en el tablero por falta de espacio.

En resumidas cuentas, el administrativo lleva al tablero, exclusivamente, la información necesaria para ejecutar una parte del trabajo. Cuando necesita más datos, por haber concluido esa parte del trabajo o porque los datos no son ya los adecuados, actualizará la información adherida al panel. Esta es una situación parangonable con la actividad y con el método de trabajo habitual en un ordenador. Cada uno de los elementos descritos tiene su homólogo en la máquina.

La *ventanilla de recepción* equivale al *órgano para entrada de información* (normalmente, el teclado) del ordenador. El *administrativo,* rodeado de sus herramientas operativas (papel, lápiz, calculadora...) gestiona el tratamiento de la información; tarea ésta encomendada a la *unidad central de proceso* del ordenador (el microprocesador, en el caso de un ordenador personal).

El *panel* tiene su reflejo en la *memoria central* del ordenador, en la que se almacenan los datos y programas en curso de ejecución. Este cuenta con el auxilio de un *archivo* de gran capacidad, al que se traslada o del que se extrae la información de trabajo: la *memoria de masa* del ordenador.

Por último, el resultado del trabajo acometido, la factura dirigida al cliente, abandona el recinto a través de la *ventanilla de salida.* Una imagen paralela al *órgano de salida* del ordenador (por ejemplo, la pantalla de visualización).

```
1 REM -------------------
2 REM ---BASE DE DATOS---
3 REM -------------------
10 DIM A$(100,10)
199 REM -----------
200 REM ---MENU---
201 REM -----------
210 LOCATE 5, 10
220 PRINT "MENU"
230 LOCATE 10, 5
240 PRINT "1...........ENTRADA DE DATOS"
250 LOCATE 12, 5
260 PRINT "2...........EDICION DE DATOS"
270 LOCATE 14, 5
280 PRINT "3...........VISUALIZACION DE DATOS"
300 LOCATE 20,5
310 INPUT "ELIJA OPCION (1-3) ",N$
320 N=VAL(N$)
330 ON N GOSUB 1000,2000,3000
340 GOTO 200
997 REM ---------------
998 REM ---CREACION---
999 REM ---------------
1000 FOR REG=1 TO 100
1010 FOR CAM=1 TO 10
1015 PRINT "REGISTRO: ";REG;" CAMPO: ";CAM
1020 INPUT B$
1025 IF B$="@R" THEN LET CAM=10: GOTO 1040
1027 IF B$="@F" THEN LET CAM=10:LET REG=100:GOTO1040
1030 LET A$(REG,CAM)=B$
1040 NEXT CAM
1050 NEXT REG
1100 RETURN
2000 REM -------------
2001 REM ---EDICION---
2002 REM -------------
2010 INPUT "REGISTRO";REG
2020 INPUT "CAMPO";CAM
2030 PRINT "REGISTRO ";REG;"  CAMPO ";CAM
2040 PRINT "CONTENIDO: ";A$(REG,CAM)
2100 INPUT B$
2104 IF B$="" THEN GOTO 2120
2106 IF B$="@F" THEN RETURN
2108 IF B$="@R" THEN LET CAM=1:LET REG=REG+1: GOTO 2030
2110 LET A$(REG, CAM)=B$
2120 LET CAM=CAM+1
2130 IF CAM=11 THEN LET CAM=1:LET REG=REG+1
2140 IF REG <101 THEN GOTO 2030
2150  PRINT "NO HAY MAS DATOS"
2160 FOR I=1 TO 1000:NEXT I
2170 RETURN
3000 REM --------------------
3001 REM ---VISUALIZACION---
3002 REM --------------------
3010 INPUT "BUSCO EN EL CAMPO NUM.";CAM
3020 INPUT "QUE DATO";D$
3030 FOR REG=1 TO 100
3040 IF A$(REG,CAM)=D$ THEN GOTO 3100
3050 NEXT REG
3060 PRINT "NO ENCUENTRO ESE DATO"
3070 INPUT "BUSCO OTRO DATO (S/N)";B$
3080 IF B$="S" THEN GOTO 3010
3090 RETURN
3100 CLS
3110 PRINT "REGISTRO NUMERO ";REG
3120 PRINT
3130 FOR CAM=1 TO 10
3140 PRINT "CAMPO ";CAM;"=",A$(REG,CAM)
3150 NEXT CAM
3200 PRINT "ENTER...........REGISTRO SIGUIENTE"
3210 PRINT "@R................NUEVO REGISTRO"
3220 PRINT "@F................FINALIZACION"
3230 INPUT "OPCION";B$
3240 IF B$="" THEN REG=REG+1:GOTO 3100
3240 IF B$="@R" THEN GOTO 3010
3240 IF B$="@F" THEN RETURN
3250 GOTO 3100
```

Después de presentar el menú, es preciso recoger la opción elegida por el usuario. Tarea que se resuelve con un nuevo comando INPUT. Para situar el punto de entrada del dato volvemos a hacer uso de LOCATE.

```
300 LOCATE 20,5
310 INPUT "ELIJA OPCION (1—3),",N$
320 N=VAL(N$)
330 ON N GOSUB 1000,2000,3000
340 GOTO 200
```

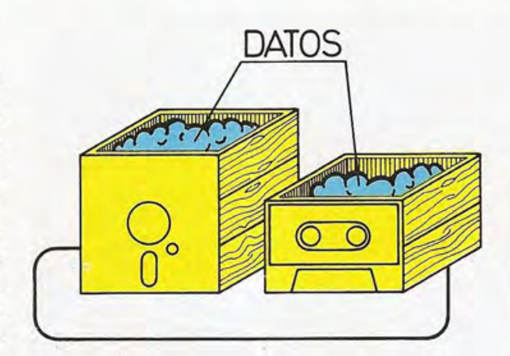
En el argumento de INPUT figura una variable de cadena. Su presencia evitará la generación del mensaje de error correspondiente a la introducción accidental de una letra. La línea 320 extrae de N$ el dato numérico, depositándolo en la variable N. Esta es la variable que se utiliza en la línea 330 para el salto a la subrutina adeucada. En el caso de no aportar un dato correcto, el GOTO de la siguiente línea hará que la secuencia de ejecución retorne a la línea 200 para visualizar de nuevo el menú de opciones.

# Archivos en BASIC (I)

## Introducción a los archivos secuenciales



El principal inconveniente que tiene un ordenador es que sólo recuerda la información durante el tiempo en que está encencido. Esto significa que al apagar el equipo se pierde la información contenida en su memoria central. Para evitar el engorro de volver a introducir los datos cada vez que se inicia una nueva sesión, hay que recurrir a la solución de almacenar la información en un soporte externo. En este punto, cabe recordar que los soportes más utilizados con los microordenadores son las cintas de casete y los discos flexibles.




*Para el almacenamiento permanente de la información se utilizan soportes de memoria conectados externamente al ordenador. En el ámbito de la microinformática, los soportes más frecuentes son la cinta de cassette y el disco magnético.*

### Archivando información

La información almacenada en un dispositivo externo se agrupa en «archivos» o «ficheros». Un archivo consiste en un bloque homogéneo de datos o instrucciones. Así, por ejemplo, cuando se almacena un programa en cinta o disco se está creando un archivo. Sin embargo, éste no es el único método para crear bloques organizados de información.

La estructura de un archivo va a depender de la forma en la que éste se almacena. En principio, un archivo se puede concebir de dos formas diferentes: secuencial o aleatorio.

Un archivo secuencial es, como su propio nombre indica, aquel al que se accede recorriendo sus elementos uno a uno (secuencialmente). Este es, precisamente, el método utilizado con las cintas en casete. En un casete es preciso pasar por los datos anteriores para acceder a un dato específico. Además, una vez leído el dato no es posible volver a él de inmediato (a no ser que se rebobine la cinta magnética manualmente). Tal es la filosofía de los archivos secuenciales.

Un archivo de tipo aleatorio o de acceso directo funciona de forma bien distinta. En él se puede acceder a un dato directamente, sin necesidad de leer los anteriores. Concretamente, la información almacenada en un archivo se agrupa en registros y dichos registros son los que pueden ser recuperados de forma directa; esto es: se puede leer el registro quinto, luego el segundo y más tarde el octavo. Por su naturaleza, este tipo de archivos no puede ser creado en casete, sino tan sólo en soportes de acceso aleatorio como es el caso del disco.

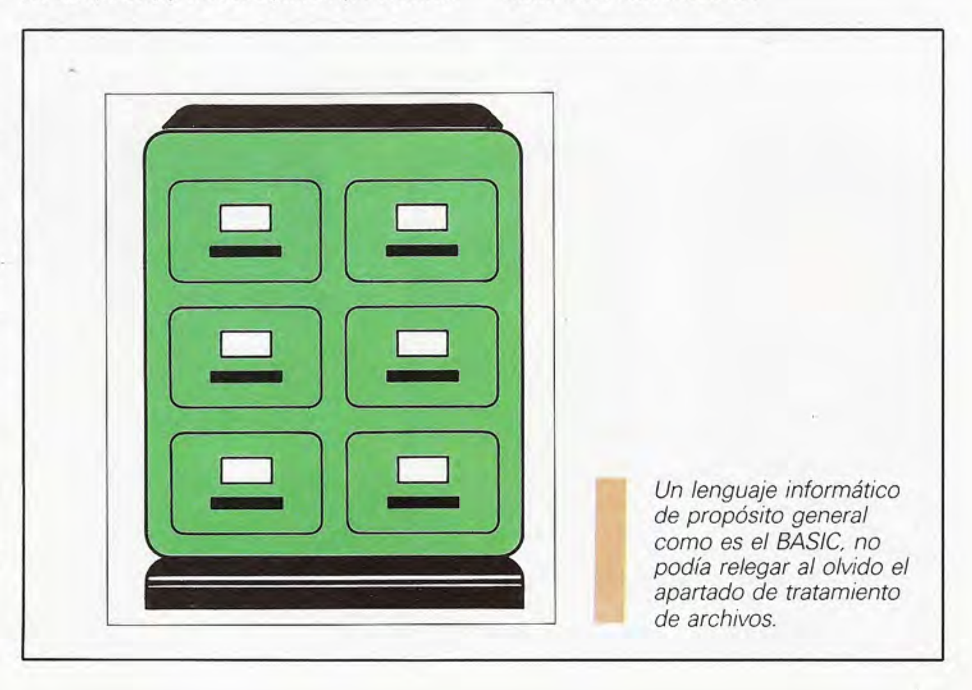
### Manejo de archivos

Para manejar los archivos es necesario seguir, paso a paso, una secuencia definida de acciones. Los pasos a dar se asemejan en cierta medida a los necesarios con un archivo físico. Esto es: apertura del archivo, consulta y cierre del mismo.

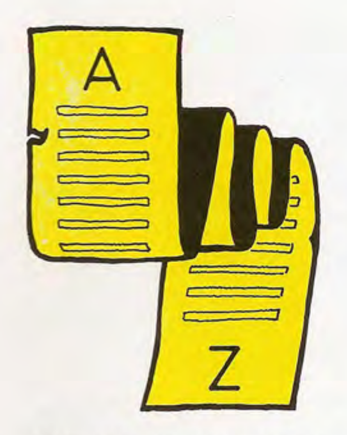
Para la apertura y cierre se hace uso de dos comandos BASIC genéricos: OPEN (abrir) y CLOSE (cerrar). El manejo de la información, sin embargo, depende del tipo de archivo.

Los archivos de tipo secuencial funcionan de modo muy semejante al teclado y a la pantalla. Para escribir en uno de ellos se mandan los datos uno tras otro, de la misma forma que los datos se mandan a la pantalla con un PRINT. La lectura se efectúa también de forma análoga, equivalente a la recogida de datos del teclado con la orden INPUT.

Si el archivo a tratar es aleatorio, habrá que tener presente el formato de los registros. Una vez definido el archivo en cuestión, la lectura y escritura se realiza trasladando a la memoria central del ordenador el registro adecuado. Con el registro en la memoria del ordenador, la lectura o escritura de sus elementos es similar a la lectura o escritura del contenido de una variable.



*Un lenguaje informático de propósito general como es el BASIC, no podía relegar al olvido el apartado de tratamiento de archivos.*




SECUENCIAL




ALEATORIO
(directo)

*Atendiendo al método de acceso a la información almacenada, cabe diferenciar entre dos tipos básicos de archivos: los secuenciales y los de acceso directo o «aleatorio».*

## Archivos secuenciales: apertura

Como se indicó anteriormente, el comando que realiza la apertura de un fichero es OPEN. Antes de proceder a la explicación de OPEN es necesario conocer algunos detalles acerca de la apertura de archivos.

Los archivos almacenados en un mismo soporte se identifican por medio de un nombre. Por ejemplo, al grabar un programa es preciso otorgar a éste un nombre que facilite su identificación. Este nombre sirve para no confundir unos archivos con otros, dentro del mismo soporte. Así pues, cada archivo ha de tener su nombre identificativo.

Como sabemos, existen dos tipos de archivos: secuenciales y aleatorios. Ambos tipos son mutuamente incompatibles, por ello es preciso especificar el tipo al acceder a un archivo. En el caso de los archivos secuenciales no es posible leer y escribir al tiempo. Ello hace que se deba optar forzosamente por una de ambas acciones.

En algunas ocasiones puede ser necesario tener abierto más de un archivo al tiempo; por ejemplo, a la hora de traspasar información de un archivo a otro, o comparar dos archivos entre sí. Si esto ocurre, será preciso tener bien diferenciados los distintos archivos abiertos.

Toda esta información debe ser indicada en la apertura del archivo. Así pues, la instrucción OPEN presenta, por lo general, el siguiente formato:

(número de línea) OPEN<modo>,#<número>,<nombre>

A continuación la palabra reservada OPEN se indica, en la zona <modo>, el tipo de archivo.

En el caso de los archivos secuenciales se especificará también si van a utilizarse como archivos de salida o de entrada, escribiendo una "O" (de Output=salida) o una "I" (de Input=entrada).

Tras el signo de número (#) se adjunta un dato o expresión numérica. Este dato identifica al archivo en cuestión entre todos los que se encuentren abiertos al mismo tiempo.

Por último, ha de figurar el nombre otorgado al fichero. Junto al nombre es corrientes indicar el dispositivo en el que se encuentra el archivo.

Este formato no es ni mucho menos generalizable, aunque sí es de los más utilizados.

Existe otro formato, también muy frecuente, el cual ofrece el siguiente aspecto:

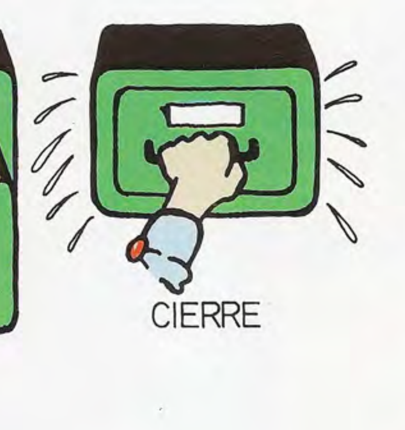
(número de línea) OPEN<nombre>FOR<modo>AS#<número>

En el que <modo> admite las palabras clave INPUT o OUTPUT.

A lo largo de este capítulo se utilizará el primero de los formatos descritos. Como ya se ha mencionado, la estandarización en este tema brilla por su ausencia. De todos modos, este capítulo sólo pretende aportar una introducción; más adelante se completará la información relativa al tema de manejo de archivos.

Con las indicaciones señaladas, la

APERTURA



CONSULTA

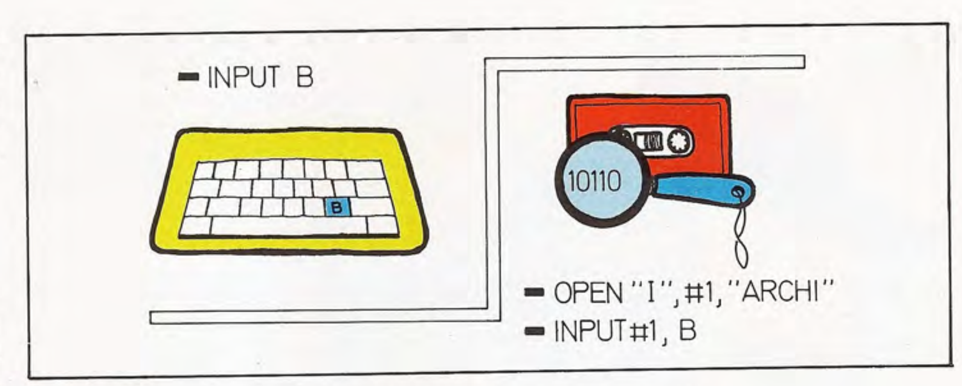
CIERRE

*Las tres operaciones básicas que hay que realizar con un archivo coinciden con las habituales en un archivo físico: apertura, consulta y cierre.*

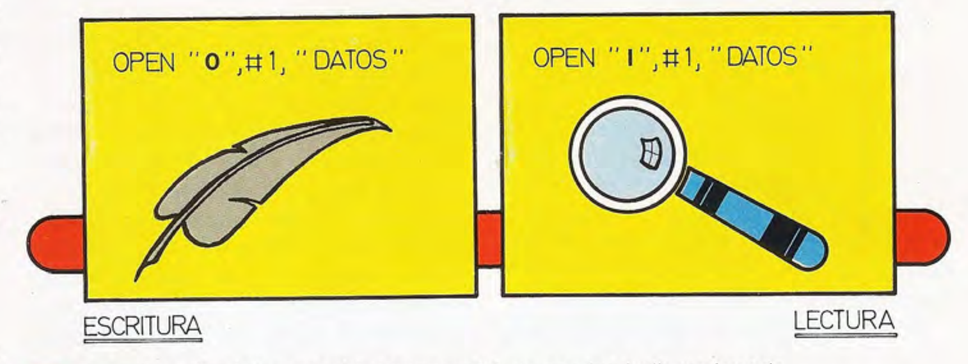
apertura de un archivo no reviste mayor complicación. Por ejemplo, para abrir un archivo de nombre DATOS, almacenado en casete, para realizar en él operaciones de lectura, bastará con introducir lo siguiente:

```
OPEN "I",#1,"C:DATOS"
```

La instrucción especifica que se realizarán operaciones de lectura sobre el archivo (entrada al ordenador) con el dato "I", y que se trata del archivo DATOS residente en casete, con el indicativo "C:DATOS". El periférico suele especificarse con algunos caracteres que preceden al nombre del archivo y separados de éste por el signo dos puntos (:). En nuestro caso hemos tomado "C:" como indicativo de la unidad de casete. Algunos equipos consideran al casete como dispositivo por defecto; esto es, si no se especifica otra cosa se supone que el archivo se encuentra en casete.




*La lectura de datos de un archivo guarda una cierta relación con la recogida de datos introducidos a través del teclado: en este caso, el comando a utilizar es INPUT.*




*A la hora de abrir un archivo secuencial, es preciso indicar el modo en el que éste se va a utilizar; éste puede coincidir con el modo escritura/ salida («O») o lectura/entrada («I»).*

Si se desea abrir un nuevo archivo para salida de datos puede utilizarse la siguiente expresión:

```
OPEN "O",#2,"D:RESULT"
```

En este segundo caso se ha utilizado la opción "O" que permite la salida de datos. El número 2 diferencia a este nuevo archivo del abierto con anterioridad. Por lo demás, se especifica que el archivo se encuentra en disco por medio del prefijo **D:**. Tras abrir el archivo puede ya procederse a su lectura o actualización.

## Escritura en un archivo secuencial

Para escribir datos en un archivo secuencial es imprescindible que éste haya sido abierto con anterioridad. En el comando de apertura se habrá otorgado al archivo un número identificador que se utilizará, posteriormente, como referencia para el acceso al archivo en cuestión. Además, el archivo se habrá abierto en modo escritura (con el carácter "O").

Como ya se mencionó anteriormente, la escritura de un archivo secuencial se asemeja a la escritura en pantalla; hasta el punto de que incluso el comando al efecto coincide: PRINT. Al actuar sobre archivos, el comando PRINT se acompaña del signo # seguido por el número del archivo a tratar.

Volviendo al ejemplo anterior, para escribir la palabra "SALIDA" en el archivo RESULT habría que teclear lo siguiente::

```
PRINT#2,"SALIDA"
```

En todo caso, lo más interesante será guardar una cantidad grande de datos, por ejemplo un «array» o matriz de datos. Para almacenar una matriz en un dispositivo externo es aconsejable utilizar un bucle FOR. El siguiente ejemplo muestra cómo realizar dicha operación.

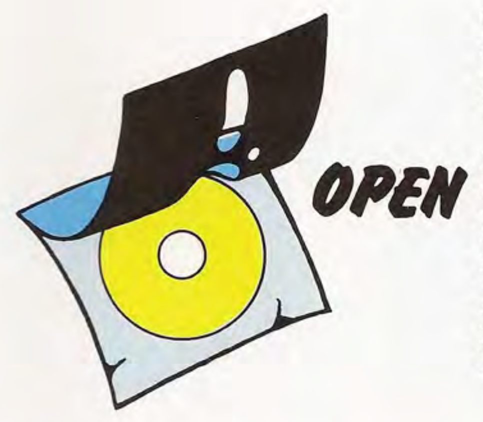
```
100 OPEN "O",#1,"C:MATRIZA"
110 FOR I=1 TO 100
120 PRINT#1,A(I)
130 NEXT I
```

### OPEN

Realiza la apertura de un archivo en un dispositivo de almacenamiento externo.

*Formato:* OPEN "<modo>",[#]<número>,<nombre>

*Ejemplos:*
```
10 OPEN "I",#1,"DATOS"
50 OPEN "O",#2,"C:ARCHIVO"
```




*OPEN es el comando BASIC especializado en la apertura de archivos.*

Estas cuatro líneas de programa almacenarán automáticamente los cien elementos de la matriz A en el archivo en casete llamado MATRIZA, datos que quedarán almacenados uno tras de otro. A la hora de proceder a su lectura éstos aparecerán en el mismo orden en el que se almacenaron, y en cualquier caso el primero en leerse será el elemento A(1).

Si se desean almacenar distintos tipos de datos en un mismo archivo será preciso utilizar varios comandos PRINT#. Un ejemplo de esta posibilidad es el que figura a continuación.

```
200 OPEN "O",#1, "LISTA"
210 PRINT#1,"LISTA DE DATOS"
220 PRINT#1,"NUMERO"
230 PRINT#1,N
240 PRINT#1,"FECHA"
250 PRINT#1,F$
260 PRINT#1,"CANTIDAD"
270 PRINT#1,TOTAL+1200
```

Los datos enviados a un archivo pueden formatearse utilizando la opción USING, al igual que en el caso de la presentación en pantalla. En este último ejemplo se podría limitar el espacio ocupado en el archivo de la siguiente forma.

```
200 OPEN "O",#1, "LISTA"
210 PRINT#1,"LISTA DE DATOS"
220 PRINT#1,"NUMERO"
230 PRINT#1,USING "###";N
240 PRINT#1,"FECHA"
250 PRINT#1,USING "  ";F$
260 PRINT#1,"CANTIDAD"
270 PRINT#1,USING "######,.#"; TOTAL+1200
```

## Cierre del archivo

Una vez realizadas las operaciones pertinentes, es conveniente cerrar de nuevo el archivo. El cierre de un archivo deja libre el número de archivo ocupado por éste. En consecuencia, se pueden abrir dos archivos con el mismo número, uno después de cerrar el otro. No obstante, el principal cometido del cierre de un archivo secuencial es marcar el final del mismo.

Cuando se graba un archivo en casete se añaden unos identificadores a la información propia del archivo. Entre estos indicadores está la cabecera, situada al principio, que guarda el nombre del archivo. La cabecera se genera a partir de la ejecución del comando OPEN.

Otro indicador es la marca de fin de archivo. Esta marca indica que el archivo no contiene más información. Con ella se evita el seguir leyendo más allá de los límites de un archivo.

El comando de cierre de un archivo es CLOSE. Para ordenar el cierre sólo es necesario conocer el número asociado al archivo. Como ya se sabe, en la apertura se aportan todos los datos identificativos del archivo y se otorga a éste un número. Es precisamente ese número el que definirá el archivo a cerrar. La formulación de este comando es, pues, la que se indica.

(número de línea) CLOSE#<número>

De esta forma, para cerrar cualquiera de los archivos creados en el apartado anterior, bastaría con introducir la siguiente orden:

```
CLOSE#1
```

El tratamiento de archivos por parte del ordenador no es tan sencillo como aparece a ojos del usuario. El ordenador ha de mandar la información a la velocidad adecuada para su grabación en la unidad de almacenamiento. Esta velocidad es generalmente inferior a la velocidad de operación del ordenador, por lo que ha de ser adaptada. Para ello, crea una zona de memoria, denominada "buffer", en la que almacena los datos a transmitir; estos datos se van mandando a la velocidad requerida por la unidad de almacenamiento. El buffer se crea por efecto del correspondiente comando de apertura OPEN. En realidad, el número asignado a cada archivo abierto no es más que el identificador del buffer asociado al mismo.

Cuando se ejecuta un comando CLOSE, al tiempo de cerrar el archivo correspondiente, se libera la zona de memoria reservada a su buffer. Por este motivo es importante cerrar los archivos una vez que han terminado las operaciones con los mismos.

### CLOSE

Cierra el archivo cuyo número figura en su argumento.

*Formato:* CLOSE[#]<número>

*Ejemplos:* 20 CLOSE#1
70 CLOSE 2

## Lectura del archivo

Hemos visto que la escritura en un archivo secuencial es análoga a la escritura en pantalla. Consecuentemente, la lectura se asemeja en gran medida a la recogida de información a partir del teclado.

La lectura de datos almacenados en un archivo secuencial se realiza por medio del comando INPUT#. Su formulación es la misma que la del INPUT normal, con la adición del número de archivo a leer.

(número de línea) INPUT#<número<, <var.1>[,<var.2<...]

Como es natural, el archivo ha de estar abierto previamente y con la opción de lectura ("I"). Dadas las características de las cintas de casete, es imposible abrir en ellas más de un archivo al tiempo (la cabeza no puede posicionarse en dos lugares a la vez). Tampoco es posible leer y escribir en un mismo archivo. Estas posibilidades quedan reservadas para cuando se dispone de varias unidades de casete o de una unidad de disco.

Para la lectura de un archivo en casete hay que situar la cinta en el punto adecuado (en cualquier posición anterior al comienzo del archivo). Hecho esto, se debe abrir el archivo o proceder a su lectura. A título de ejemplo, veamos los comandos capaces de leer la matriz almacenada en el apartado anterior.

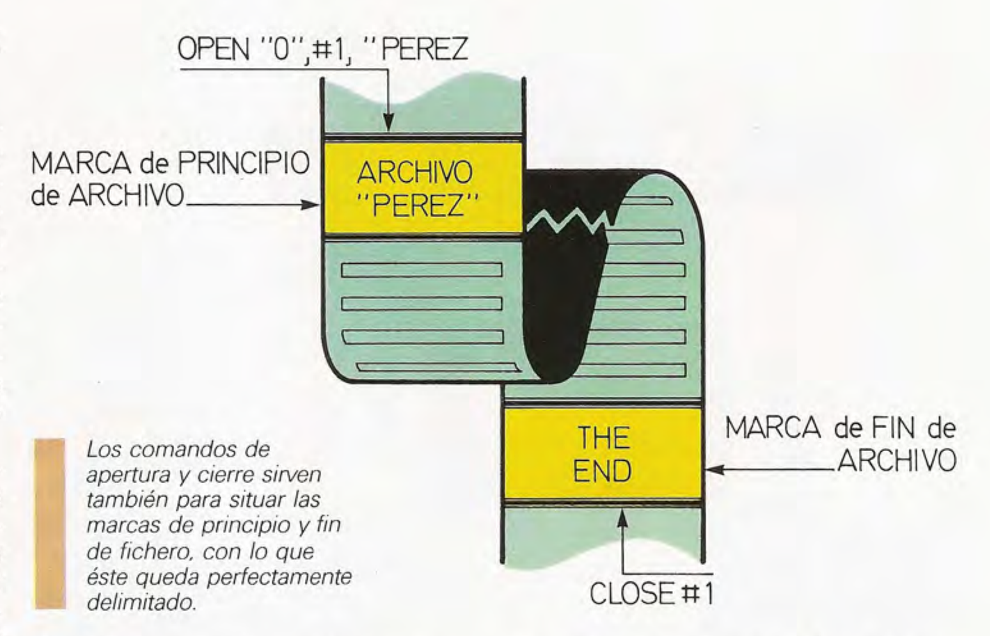
```
1000 OPEN "I",#1,"C:MATRIZA"
1010 FOR I=1 TO 100
1020 INPUT#1,B(I)
1030 NEXT I
1040 CLOSE#1
```

Cuando se trata de archivos, el comando INPUT no permite la opción de presentación de un mensaje. Así pues, la orden INPUT#1 "DATO", D produciría un error de tipo sintáctico.

Como se ha visto en el anterior ejemplo, el empleo de INPUT# es tan sencillo como el de PRINT#, el único cuidado está en no olvidar la apertura y cierre del archivo. Para completar la serie de ejemplos veamos cómo se realizaría la lectura del otro archivo creado en un apartado precedente:

```
2000 OPEN "I",#1"LISTA"
2010 INPUT#1,CABE$
2020 INPUT#1,A$
2030 INPUT#1,NUM
2040 INPUT#1,B$
2050 INPUT#1,DIA$
2060 INPUT#1,C$
2070 INPUT#1,TOTAL
```

Como se observa, no es preciso que los nombres de las variables coincidan con los que se utilizaron al crear el archivo. Sin embargo, los tipos de las variables sí deben estar en correspondencia.



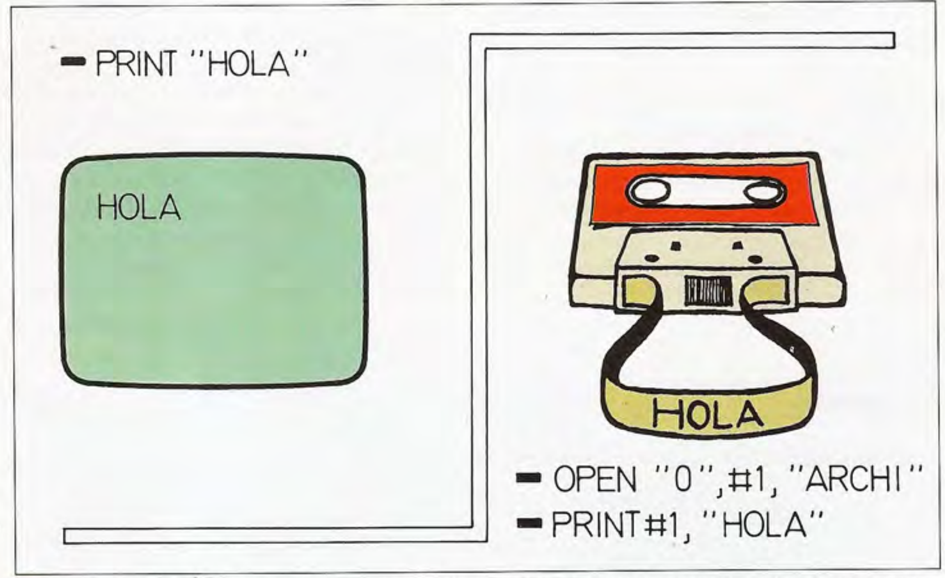

*Los comandos de apertura y cierre sirven también para situar las marcas de principio y fin de fichero, con lo que éste queda perfectamente delimitado.*

## Archivos como bases de datos

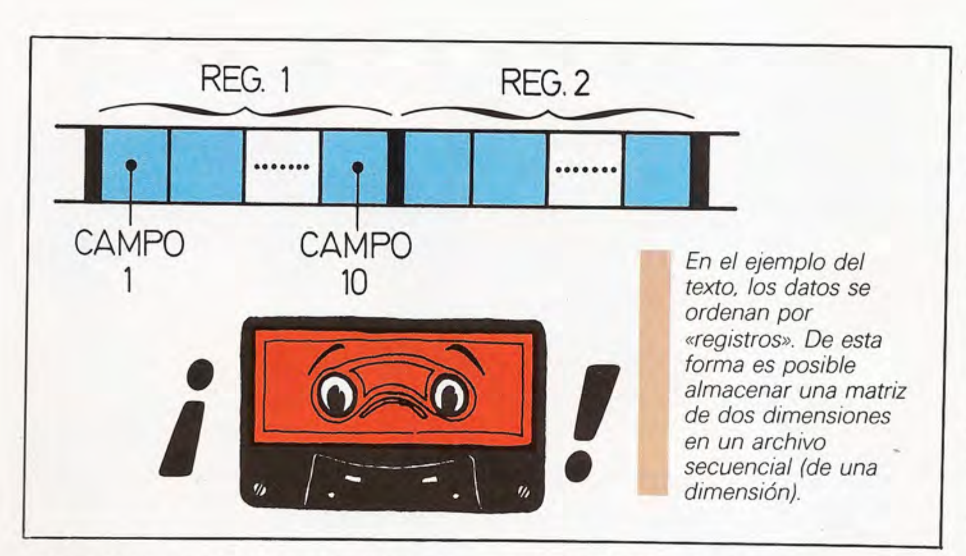
En el capítulo anterior se describió un ejemplo consistente en el diseño de una base de datos. El programa así creado puede resultar de gran utilidad; sin embargo, es preciso introducir de nuevo los datos cada vez que se desee utilizar el programa. Para solventar esta situación puede hacerse uso de los archivos recién presentados.

En aquel ejemplo se utilizaba una matriz de 100×10 para almacenar los datos. El paso de esta matriz desde la memoria central al casete es muy sencillo. Al tratarse de una matriz de dos dimensiones, son necesarios dos bucles anidados. Una primera aproximación a la rutina de almacenamiento la constituyen las siguientes líneas.

```
4110 OPEN "0",#1,"BASE"
4120 FOR REG=1 TO 100
4130 FOR CAM=1 TO 10
4140 PRINT#1,A(REG,CAM)
4150 NEXT CAM
4160 NEXT REG
4170 CLOSE#1
```




*Para escribir en un archivo secuencial se actúa de forma parecida a la escritura en pantalla. El comando utilizado a tal efecto es el conocido PRINT.*




*En el ejemplo del texto, los datos se ordenan por «registros». De esta forma es posible almacenar una matriz de dos dimensiones en un archivo secuencial (de una dimensión).*

Los dos bucles harán que la matriz se grabe registro a registro. Dentro de cada registro se almacenan los campos por su orden. Aunque también es posible grabar la matriz en otro orden: por campos dentro de los cuales los registros se mantengan ordenados. Para utilizar esta otra distribución basta con cambiar de sitio los bucles.

```
4110 OPEN "0",#1"BASE"
4120 FOR CAM=1 TO 10
4130 FOR REG=1 TO 100
4140 PRINT#1,A(REG,CAM)
4150 NEXT REG
4160 NEXT CAM
4170 CLOSE#1
```

El orden en que se almacenan los datos no importa, siempre que se emplee ese mismo orden a la hora de proceder a su lectura. En todo caso, y dado que es el más intuitivo, vamos a poner en práctica el primer método.

En las líneas escritas hasta ahora se realiza correctamente el almacenamiento de datos. No obstante, el archivo en el que se guardan es siempre el mismo: "BASE". Lo más lógico sería poder diferenciar distintos conjuntos de datos dando diferentes nombres a los archivos. Esto es perfectamente posible ya que el comando OPEN admite variables en su argumento. De esta forma se puede elegir el nombre del archivo con entera libertad.

```
4010 INPUT "NOMBRE DEL ARCHIVO";N$
4110 OPEN "0",#1,N$
.......................
.......................
```

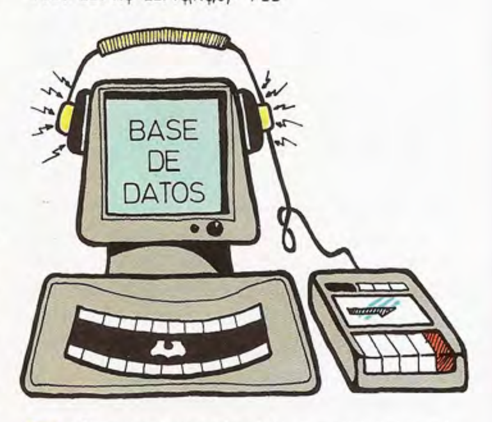
En estas líneas se recoge el nombre y se abre el archivo cuyo nombre coincide con el indicado. La longitud del nombre vendrá limitada por el equipo en cuestión, y más concretamente por el sistema operativo. La longitud de los nombres de archivo suele situarse alrededor de los ocho caracteres. Cuando se trabaja con unidad de disco depende del sistema operativo empleado, los habituales CP/M y MS-DOS utilizan ocho caracteres más tres de «extensión».

Con la posibilidad de elegir el nombre del archivo se abren otras ventajas interesantes. Por ejemplo, si usted guarda los archivos de la base de datos en una cinta donde residan más tipos de archivos, querrá poder distinguirlos. Para diferenciar los archivos creados por la base de datos puede concluir su nombre con algunas letras características, por ejemplo BD (Base de Datos). Ello se consigue con el tratamiento del dato de cadena introducido en la línea 4010.

```
4010 INPUT "NOMBRE DEL ARCHIVO";N$
4020 LET N$=N$+"/BD"
4110 OPEN "0",#1,N$
.......................
.......................
```

En el proceso de la línea 4020 (adición de los caracteres "/", "B"y "D") puede rebasarse la longitud máxima especificada para el nombre. Si el límite está en ocho caracteres, habrá que elegir cinco de los aportados por el usuario, que sumados a los tres añadidos, hacen en total ocho. Esta sería la nueva línea:

```
4020 LET N$=LEFT$(N$,5)+"/BD"
```




*En una base de datos —como es el caso del ejemplo propuesto en el capítulo anterior—, resulta fundamental la posibilidad de emplear archivos residentes en una unidad de almacenamiento externo.*

Con lo expuesto, la rutina de escritura del archivo presentará un nuevo aspecto:

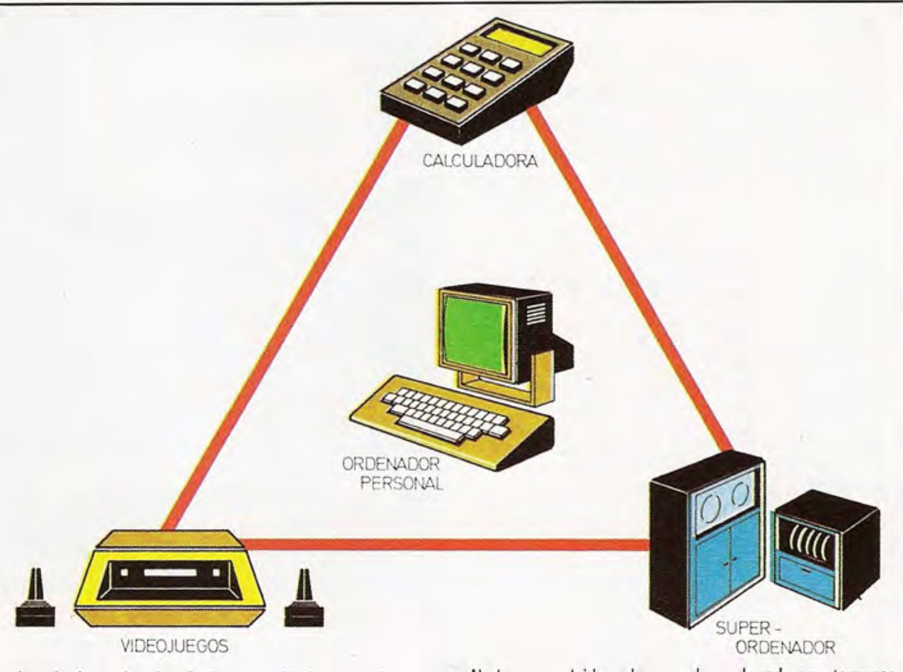
```
4000 REM     ------------------------------
4001 REM     -------------GRABACION--------
4002 REM     ------------------------------
4010 INPUT "NOMBRE DEL ARCHIVO";N$
4020 LET N$=LEFT$(N$5)+"/BD"
4110 OPEN "0",#1,N$
4120 FOR REG=1 TO 100
4130 FOR CAM=1 TO 10
4140 PRINT#1,A(REG,CAM)
4150 NEXT CAM
4160 NEXT REG
4170 CLOSE#1
4180 RETURN
```

## PRINT#

Escribe un dato en el archivo especificado.

*Formato:* PRINT# <número>, <dato> [;<dato>...]
*Ejemplos:* 20 PRINT #1, "PERICO"
70 PRINT #2, A+B

## ¿Qué es y qué no es un ordenador personal?




Cualquier ordenador, sea cual fuere su tamaño y potencia, es un producto de la síntesis de dos elementos complementarios: un soporte físico o circuito electrónico, el *«hardware», y una programación o conjunto de instrucciones, datos, programas..., el «software»*. Ambos elementos se conjugan en un sistema para el tratamiento de información u ordenador. Aquí aparece la distinción esencial entre el ordenador y otras máquinas capaces sólo de resolver un determinado número de tareas específicas (una calculadora, por ejemplo). El ordenador es un sistema cuya funcionalidad no está predefinida por su estructura física, sino que puede ser «instruido» por el usuario para realizar una u otra función introduciéndole un programa al efecto.

Una calculadora convencional es capaz de realizar ciertas operaciones matemáticas (suma, resta, multiplicación, división...), pero única y exclusivamente esas operaciones preestablecidas. Por su parte, el ordenador posee un campo de aplicación totalmente versátil, definible en cualquier instante por medio del adecuado programa. Puede realizar cálculos complejos en los que intervengan secuencias de operaciones, comparaciones y decisiones, y su efectividad no se limita a los cálculos matemáticos, sino que se extiende a cualquier aplicación definible como tratamiento de información sea ésta de naturaleza simplemente numérica o alfanumérica.

La diferencia del ordenador con otras máquinas más sofisticadas y que, en muchos casos, incorporan en su interior un microprocesador, es ya más sutil. Las consolas de video-juegos constituyen un perfecto ejemplo de equipo dotado de una unidad central de proceso integrada (un microprocesador), y capaz de ejecutar un programa (el cartucho de juego). En este caso, la distinción básica reside en que tales máquinas siguen encerradas dentro de un marco de aplicación específico: la ejecución de juegos sobre una pantalla. No están abiertas a lenguajes de programación que versatilicen sus posibilidades, y tampoco disponen de un sistema operativo que ponga toda su potencialidad en manos del usuario.

Una vez delimitado el terreno de los ordenadores o máquinas programables para el tratamiento de la información, llega el momento de caracterizar al ordenador personal.

No hay que olvidar a los grandes ordenadores y tampoco a los miniordenadores, e incluso a los microordenadores evolucionados. Aunque cada vez son mayores las intersecciones entre las diversas categorías de ordenadores, puede enmarcarse al ordenador personal en una zona propia. Dentro de un ámbito delimitado por su definición más amplia: máquina programable basada en microprocesador, destinada al tratamiento de información y orientada al usuario individual; con una gama de periféricos, sistemas operativos, lenguajes de programación y programas de aplicación concebidos específicamente para su explotación.

## Lectura de los archivos de datos

La contrapartida a la grabación es la lectura de los archivos. La zona de lectura de los datos es idéntica a la de la escritura, sin más que cambiar las órdenes PRINT por INPUT. Esto es:

```
5120 FOR REG=1 TO 100
5130 FOR CAM=1 TO 10
5140 INPUT#1,A(REG,CAM)
5150 NEXT CAM
5160 NEXT REG
```

No hay que olvidar que el orden de lectura debe ser congruente con el de escritura. Si usted lo alteró en el apartado anterior, debe tomar la misma medida en éste. La zona de apertura tendrá que construir el nombre de la misma forma que lo hacía la de escritura. La única diferencia reside en la presencia del carácter "I" que indica que se trata de un archivo de entrada.

```
5010 INPUT "NOMBRE DEL ARCHIVO";N$
5020 LET N$=LEFT$(N$,5)+"/BD"
5110 OPEN "I",#1,N$
```

Por último, han de introducirse las líneas de cierre del archivo y retorno al programa principal. Estas dos líneas sí son exactamente iguales a las empleadas en la rutina anterior.

### INPUT#

Lee el siguiente dato almacenado en el archivo que se indica en su argumento.

*Formato:* INPUT#<número>,<var>[;<var>...]

*Ejemplos:* 20 INPUT#1,A$
70 INPUT#3,DATO

```
5170 CLOSE#1
5180 RETURN
```

Así pues, siguiendo los pasos comentados, se llega a la consecución de la rutina de lectura. Esta rutina completa y con los REM de identificación es la que sigue:

```
5000 REM     ——————————————
5001 REM     ——— LECTURA ———
5002 REM     ——————————————
5010 INPUT "NOMBRE DEL ARCHIVO";N$
5020 LET N$=LEFT$(N$,5)+"/BD"
5110 OPEN "I",#1,N$
5120 FOR REG=1 TO 100
5130 FOR CAM=1 TO 10
5140 INPUT#1,A(REG,CAM)
5150 NEXT CAM
5160 NEXT REG
5170 CLOSE#1
5180 RETURN
```

## El último detalle

Para el correcto funcionamiento de estas rutinas con el programa analizado en el capítulo anterior, quedan por añadir unos retoques al programa original. En primer lugar, las opciones de lectura y escritura del archivo deben estar contempladas en el menú inicial. Para ello, habrá que introducir nuevas líneas que escriban sus nombres en pantalla.

```
285 LOCATE 16,5
288 PRINT "4........GRABACION DEL ARCHIVO"
```




*Las dos nuevas opciones introducidas han de verse contempladas en el menú. Unos ligeros retoques en la correspondiente rutina servirán a tal propósito.*

```
290 LOCATE 18,5
295 PRINT "5........LECTURA DE ARCHIVO"
```

Con estas líneas se obtiene la presentación en pantalla de ambas posibilidades. Pero con todo esto no basta, también es necesario contemplar dichas opciones en el salto a las subrutinas; esto es: hay que cambiar la línea del ON/GOSUB. Como las nuevas rutinas se encuentran localizadas a partir de las líneas 4000 y 5000, la modificación es la siguiente:

```
330 ON N GOSUB 1000,2000,3000,4000,5000
```

Como último retoque hay que modificar ligeramente la línea 310, que contiene el comando INPUT. La línea original muestra el siguiente aspecto.

```
310 INPUT "ELJA OPCION (1-3)",N$
```

Esta línea, después de la modificación, debe indicar que existen cinco posibles opciones. Así pues, la nueva línea quedará como figura a continuación.

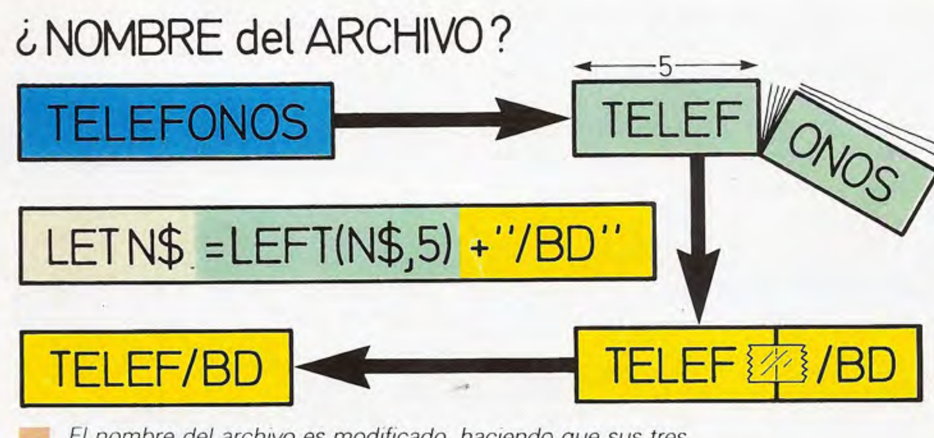
```
310 INPUT "ELJA OPCION (1-5)",N$
```

Una vez incluidas las modificaciones señaladas no encontraremos ante la siguiente rutina:

```
199 REM     ————————
200 REM     —— MENU ——
201 REM     ————————
210 LOCATE 5,10
220 PRINT "MENU"
230 LOCATE 10,5
240 PRINT "1........ENTRADA DE DATOS"
250 LOCATE 12,5
260 PRINT "2........EDICION DE DATOS"
270 LOCATE 14,5
280 PRINT "3........VISUALIZACION DE DATOS"
285 LOCATE 16,5
288 PRINT "4........GRABACION DEL ARCHIVO"
290 LOCATE 18,5
295 PRINT "5........LECTURA DEL ARCHIVO"
300 LOCATE 20,5
310 INPUT "ELJA OPCION (1-5)",N$
320 N=VAL(N$)
330 ON N GOSUB 1000,2000,3000,4000,5000
340 GOTO 200
```

Introduciendo los cambios, que se resumen en las rutinas de lectura y escritura del archivo y la de menú, el programa del capítulo anterior resulta verdaderamente operativo. En realidad, en una base de datos es fundamental mantener los archivos en un dispositivo de almacenamiento externo.

En la apertura de los archivos se ha prescindido de la especificación de dispositivo. Tal como se ha programado se utilizará el dispositivo que el ordenador tome «por defecto»; en la mayor parte de los casos, éste corresponde a la unidad de casete. Para cambiar de dispositivo bastaría con indicarlo en la apertura de los archivos.




*El nombre del archivo es modificado, haciendo que sus tres últimos caracteres sean BD. Con ello, se ve facilitada la identificación de los archivos creados por medio de este programa.*

## TABLA DE CONVERSION

| Ordenador | Apertura | | Cierre | Escritura | Lectura | Soporte de memoria |
|---|---|---|---|---|---|---|
| | OPEN "<modo>", #<n.º>,<nom> | OPEN <nom> FOR <MAS<n.º> | CLOSE#<n.º> | PRINT#<n.º>, <dato> | INPUT#<n.º>, <var> | |
| AMSTRAD | OPENIN/ OPENOUT (8) | — | CLOSEIN/ CLOSEOUT (8) | PRINT #9, <dato> | INPUT#9, <dato> | Disco/Casete |
| APPLE II (APPLESOFT) | OPEN <nombre> (1) | — | CLOSE <nombre> | WRITE <nombre> (2) | READ <nombre> (2) | Disco |
| APRICOT (M-BASIC) | OPEN "<modo>", #<n.º>,<nom> | OPEN <nom> FOR <m>AS<n.º> | CLOSE#<n.º> | PRINT#<n.º>, <dato> | INPUT#<n.º>, <var> | Disco |
| ATARI | OPEN#<n.º>, <modo>,<nom> (3) | — | CLOSE#<n.º> | PRINT#<n.º>, <dato> | INPUT#<n.º>, <var> | Casete |
| CBM 64 | OPEN "<n.º>", 1,<modo><nom> | — | CLOSE#<n.º> | PRINT#<n.º>, <dato> | INPUT#<n.º>, <var> | Casete |
| DRAGON | OPEN "<modo>", #-1,<nom> | — | CLOSE#-1 | PRINT#-1, <dato> | INPUT#-1, <var> | Casete |
| EQUIPOS MSX | — | OPEN <nom> FOR <m>AS<n.º> | CLOSE#<n.º> | PRINT#<n.º>, <dato> | INPUT#<n.º>, <var> | Casete |
| HP-150 | OPEN "<modo>", #<n.º>,<nom> | OPEN <nom> FOR <m>AS<n.º> | CLOSE#<n.º> | PRINT#<n.º>, <dato> | INPUT#<n.º>, <var> | Disco |
| IBM PC | OPEN "<modo>",# <n.º>,<nom> | OPEN <nom> FOR <m>AS<n.º> | CLOSE#<n.º> | PRINT#<n.º>, <dato> | INPUT#<n.º>, <var> | Disco |
| MPF | — | — | — | — | — | — |
| NCR DM-V (MS-BASIC) | OPEN "<modo>", #<n.º>,<nom> | — | CLOSE#<n.º> | PRINT#<n.º>, <dato> | INPUT#<n.º>, <var> | Disco |
| NEW BRAIN | OPEN <modo>,# <n.º>,1,<nom> | — | CLOSE#<n.º> | PRINT#<n.º>, <dato> | INPUT#<n.º>, <var> | Casete |
| ORIC | — | — | — | — | — | — |
| SHARP MZ-700 (MZ-BASIC) | — | — | — | — | — | — |
| SINCLAIR QL | OPEN#<n.º>, <nom> (6) | — | CLOSE#<n.º> | PRINT#<n.º>, <dato> | INPUT#<n.º>, <var> | Microdrive |
| SPECTRAVIDEO | — | OPEN <nom> FOR <m>AS<n.º> | CLOSE#<n.º> | PRINT#<n.º>, <dato> | INPUT#<n.º>, <var> | Casete |
| ZX-SPECTRUM | — | — | — | SAVE <nom> DATA <Array> (7) | LOAD <nom> DATA <Array> (7) | Casete |

Las formulaciones reflejadas en la tabla para cada ordenador, corresponden al caso del soporte de memoria que se especifica en la columna correspondiente.

(1) En Apple, los comandos de manejos de archivos se introducen como cadenas de caracteres, precedidas de un CTRL-D, dentro de instrucciones PRINT. Por ejemplo:
10 D$=" ":REM CTRL-D
20 PRINT D$"OPEN DATOS"

(2) Los comandos WRITE y READ sirven para que los siguientes PRINT o INPUT no actúen con la pantalla y el teclado, sino con el archivo indicado.

(3) En la zona <modo> se utilizan los números 4 para entrada y 8 para salida de datos.

(4) El modo se especifica como 0 (lectura) ó 1 (escritura).

(5) En <modo> se emplea IN en lugar de "I" y OUT en lugar de "O". El número 1 indica el primer conector de casete; se puede utilizar el segundo poniendo un 2.

(6) El nombre de archivo en microdrive debe tener la forma MDV<n.º>-<nombre>, siendo <n.º> el número de la unidad de mocrodrive. Por ejemplo, MDV1-ARCHI.

(7) Se utilizan para la grabación y lectura de ARRAYS en casete.

(8) Para más información consult

## Operadores lógicos

En BASIC existe un tipo de dato, denominado «lógico», cuya presencia es harto frecuente en el argumento del comando IF. En un capítulo precedente se vio que este comando era capaz de evaluar condiciones, entregando un resultado que podía ser «verdadero» o «falso». Pues bien, los valores que puede adoptar un dato de tipo lógico son precisamente los de «verdadero» y «falso». Los datos lógicos se almacenan en el ordenador a modo de datos numéricos. Por regla general, se asocia al valor de «cierto» el dato numérico -1 y al de «falso» el 0. De esta forma se hace posible el almacenamiento de un valor lógico en el seno de una variable numérica normal.

( ) PARENTESIS
— SIGNO MENOS
^↑ EXPONENCIACION
✱/ MULTIPLICACION Y DIVISION
+ — SUMA Y RESTA
= < > <> ≤ ≥ OPERADORES DE RELACION
NOT COMPLEMENTO LOGICO
AND Y LOGICO
OR O LOGICO
XOR O-EXCLUSIVO

*Jerarquía de las operaciones aritméticas, lógicas y de relación programables en BASIC.*

Los datos lógicos se obtienen a partir de la evaluación de condiciones, o como resultado de ciertas funciones, y su empleo más común se encuentra en el control del flujo de ejecución (habitualmente a través de un comando IF). En todo caso, el uso de datos lógicos no se limita a este cometido; también se puede operar con datos de este tipo, de forma parecida a la operación con datos numéricos o alfanuméricos.

Así pues, dos o más datos lógicos puden ser agrupados para producir un resultado también de tipo lógico. Para entender el significado de los operadores lógicos es preciso introducir algunos conceptos generales de lógica.

Tras evaluar una proposición puede traducirse que la misma es «verdadera» o «falsa». Por proposiciones se entiende cualquier tipo de expresión que afirma un hecho. Por ejemplo, «Madrid es la capital de Italia» es una proposición falsa.

Como ya se ha indicado, pueden agruparse más de una proposición sencilla para obtener una proposición compuesta. Así: «Madrid es la capital de Italia o Madrid es la capital de España» es una proposición compuesta por las simples: «Madrid es la capital de Italia» y «Madrid es la capital de España». En este caso, el nexo de unión de ambas se materializa en la palabra «o». La proposición compuesta puede ser evaluada a partir del resultado de las simples. En el ejemplo, se tiene una proposición falsa y otra verdadera. Intuitivamente, se deduce que la proposición compuesta será verdadera cuando lo sea una de las simples (cuando lo sea la primera **o** lo sea la segunda). Es decir, basta que una de ellas sea cierta para que el total sea verdadero. Ello se debe al uso de la conjunción «o». Por el contrario, si se unen las proposiciones con la conjunción «y» la cosa es bien distinta. La unión por medio de «y» determina que la veracidad de la proposición compuesta tiene lugar únicamente cuando ambas son ciertas (cuando lo es la primera **y** la segunda). Generalizando, se puede decir que las palabras **o** e **y** son dos operadores lógicos.

Para identificar con claridad el resultado de cada uno de estos operadores, se hace uso de las denominadas tablas de verdad.

Una tabla de verdad no es más que la representación de los resultados de cada operador para todas las combinaciones posibles de sus operarios. A modo de ejemplo, junto al texto se incluyen las tablas de verdad de los comentados operadores lógicos. En ellas se reflejan los resultados de operar los valores de entrada situados en las dos primeras columnas.

Existe un tercer operador lógico que corresponde a «negación». Negar una proposición supone invertir su valor lógico: la negación de «hoy es jueves» será, pues, «hoy **no** es jueves». A la vista del ejemplo se observan dos cosas: el operador «**no**» actúa sobre un único operando, y su función es la de cambiar verdadero por falso y viceversa. Su tabla de verdad coincide con la que parece al margen,

Los operadores lógicos del BASIC se formulan recurriendo a la traducción inglesa de las mismas palabras españolas: AND (y), OR (o) y NOT (no). Por lo demás, funcionan tal y como se ha descrito más arriba. El resultado de una operación lógica será un dato lógico y, por lo tanto, adecuado para el uso en el argumento de un comando IF.

Veamos a continuación un ejemplo de su uso dentro de un programa BASIC. El objetivo del programa es mostrar un mensaje de error cuando sea tecleado un número comprendido entre 10 y 20. El fragmento de programa que realiza tal acción es el siguiente:

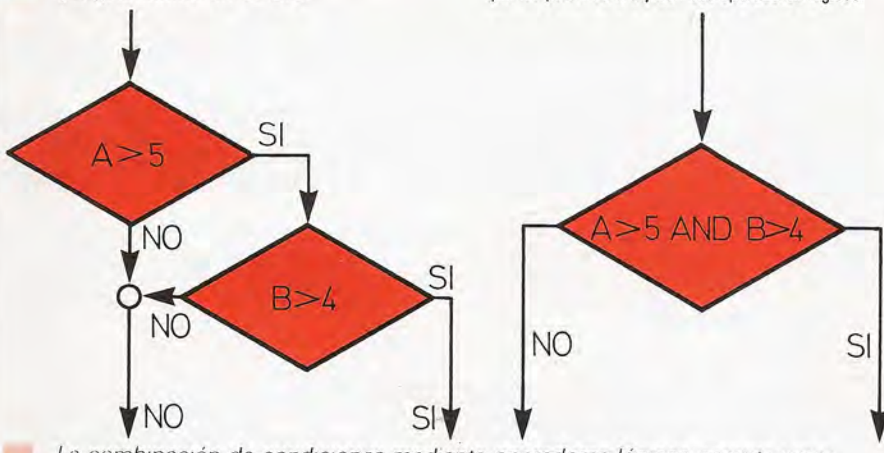
```
...
100 INPUT A
110 IF (A<20) AND (A> 10) THEN PRINT "ESE
    NUMERO NO VALE"
...
```

En el argumento se ha formulado una proposición compuesta por A<20 y A>10. Como en este caso ambas condiciones se han unido por medio del operador AND (y), la proposición conjunta sólo será verdadera cuando se cumplan los dos términos: A menor que 20 **y** mayor que 10.

Al igual que ocurre con los operadores aritméticos (+, -, *, etc.), los operadores lógicos pueden encadenarse para construir expresiones complejas. Por ejemplo, si a los números erróneos del ejemplo anterior se desean unir también los comprendidos entre 50 y 80, bastaría con introducir lo siguiente:

```
...
100 INPUT A
110 IF (A<20 AND A>10) OR (A<80 AND A>50) THEN
    PRINT "ESE NUMERO NO VALE"
...
```

En este segundo ejemplo se han utilizado los resultados de los dos AND como operandos de la relación OR. Con ello se consigue la visualización del mensaje cuando se cumpla la primera **o** la segunda de las condiciones.



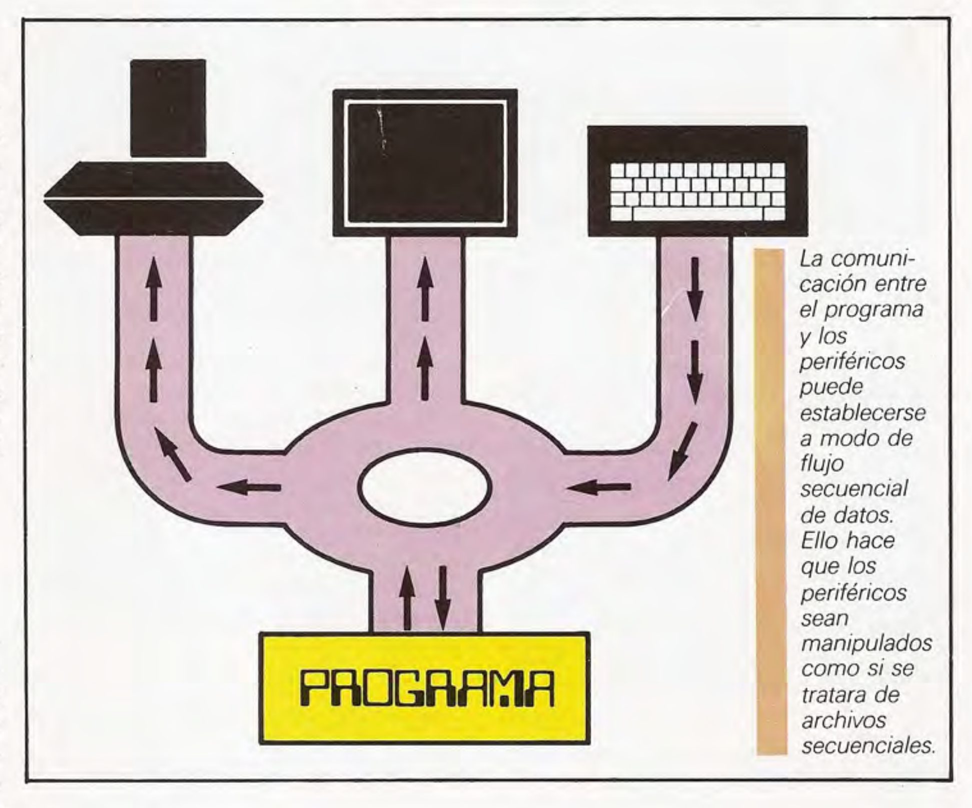

*La combinación de condiciones mediante operadores lógicos permite crear estructuras de decisión evolucionadas, definiendo condiciones múltiples en una sola línea de programa.*

# Archivos en BASIC (II)

## Uso eficiente de los archivos secuenciales



En un capítulo precedente se introdujo el concepto de archivo y se señalaron las principales características de esta estructura de almacenamiento. En aquella ocasión se estudiaron los comandos básicos destinados al tratamiento de archivos secuenciales. El objeto de este capítulo es profundizar en el tema, tratando de obtener un mayor partido del uso de este tipo de archivos. Para ello, se presentarán las restantes herramientas al efecto que aporta el lenguaje BASIC.




*La comunicación entre el programa y los periféricos puede establecerse a modo de flujo secuencial de datos. Ello hace que los periféricos sean manipulados como si se tratara de archivos secuenciales.*

### Variables de control

Antes de seguir adelante es conveniente repasar la filosofía del tratamiento y manipulación de archivos. Desde el punto de vista del programador, existen tres etapas básicas en el trabajo con archivos. Estas se identifican como: apertura, consulta y cierre. Las operaciones de apertura y cierre se efectúan por medio de los comandos OPEN y CLOSE, respectivamente.

En cuanto a la consulta, ésta puede conducir a la adición o recuperación de datos. La dirección de flujo de datos, ya sea entrada (desde el archivo) o salida (hacia el mismo), ha de ser indicada en la propia instrucción de apertura. Dicho flujo se controla empleando los comandos PRINT# (salida) e INPUT# (entrada).

Un ejemplo práctico del uso de un archivo secuencial consiste en el almacenamiento de una matriz.

```
...
100 OPEN "O",#1,"MATRIZ"
110 FOR I=1 TO 20
120 PRINT# 1,A(I)
130 NEXT I
140 CLOSE#1
```

Para recuperar los datos es suficiente con formular la rutina inversa. Esto es: sustituir el comando PRINT# por INPUT# y cambiar el modo de apertura para que se autorice la entrada de datos. La rutina de lectura del archivo recién creado será, pues, la siguiente:

```
...
200 OPEN "O",#1,"MATRIZ"
210 FOR I=1 TO 20
220 PRINT# 1,B(I)
230 NEXT I
240 CLOSE#1
...
```

El proceso no reviste mayor complicación ya que, en este caso, el tamaño de la matriz es conocido. El problema aparece cuando se lee un archivo del que se desconoce la longitud. Si el archivo guarda un total de 10 datos, se producirá un error al intentar recuperar el undécimo. Ese error hará que se detenga la ejecución, con los inconvenientes que ello conlleva.

Así pues, lo más acertado es tener una indicación de la longitud del archivo que se está tratando. Esta indicación llega de la mano de una nueva función BASIC. Se trata de LOF (Length Of File =longitud del archivo) que proporciona el número de bytes de que consta un determinado archivo. LOF admite un dato de entrada en su argumento que indica el archivo del que se desea averiguar la longitud; éste ha de corresponder con el número de identificación especificado en la apertura del fichero. Véase un ejemplo.

```
...
170 OPEN "I",#1,"ARCHIV"
180 L=LOF(1)
...
```

En el ejemplo se almacena dicha longitud en la variable L. El número identificativo del archivo (en este caso el 1) es el que figura en el argumento de LOF. Tras la ejecución de estas líneas, la variable L pasa a memorizar el número de bytes de que consta el archivo "ARCHIV".

Sin embargo, esta medida no es siempre la más apropiada. También se puede calcular el número de datos contenidos en un archivo a partir del número de bytes del mismo. Pero sólo se podrá hacer de forma sencilla si todos los datos almacenados son del mismo tipo.

Existe otra función que resulta más útil para delimitar la longitud de un archivo; una función lógica que toma el valor CIERTO cuando se alcanza el fin del archivo. La palabra del BASIC asociada a la referida función es EOF (End Of File=fin de archivo). La formulación de EOF es análoga a la de LOF. Veamos un ejemplo en el que se emplea esta última función:

## LOF

Proporciona el número de bytes de que consta un determinado archivo.

*Formato:* LOF (<número de archivo>)

*Ejemplos:* 10 LET LONG=LOF(2)
50 PRINT LOF(1)/8

```
200 OPEN "I",#1,"DATOS"
210 I=1
220 IF EOF(1) THEN GOTO 260
230 INPUT# 1,B(I)
240 I=I+1
250 GOTO 220
260 CLOSE#1
...
█
```

En la rutina se emplea un bucle, controlado por la variable I, para leer los datos del archivo. La variable de control se inicializa al valor 1 en la línea 210. Tras recuperar un dato (línea 230) se incrementa dicho contador. Esto hará que los datos leídos se almacenen en posiciones sucesivas de la matriz B.

La línea 250 cierra el bucle, permitiendo la lectura del siguiente dato; sin embargo, la clave de la rutina se encuentra en la línea 220. El comando IF contenido en la misma proporciona el punto de salida del bucle. Esta se producirá cuando EOF(1) sea cierto; esto es: cuando se haya alcanzado el fin del archivo. De esta forma se evita la lectura del siguiente dato (dato, por otra parte, inexistente).

### Delimitadores de datos

En el apartado precedente se ha descrito el método a seguir para averiguar la longitud de un archivo. Como ya se comentó en el primero de los capítulos dedicados al tema de archivos, el propio ordenador se encarga de grabar una marca de fin de archivo tras el último dato. Sin embargo, todavía no se ha mencionado cómo se separan los datos entre sí. Ello va a depender de la forma en la que se hayan almacenado.

Según lo explicado hasta ahora, la grabación de datos se efectúa por medio del comando PRINT#. Dicho comando actúa de forma análoga al encargado de la presentación de datos en pantalla (PRINT, sin "#"). PRINT# seguido por un solo dato escribe dicho dato en el archivo añadiendo un carácter al final; carácter que equivale al de «retorno de carro» que se introduce al escribir en pantalla. Si se utiliza más de un dato en su argumento, éstos deben separarse por medio del signo punto y coma. Al igual que ocurre en la presentación en pantalla con PRINT, los datos se almacenan de forma contigua; sin ningún carácter entre ellos. Esto hace que no sea posible identificar los límites de cada dato. A continuación se muestra un ejemplo de lo indicado.

```
100 A$="MARI"
110 B$="CARMEN"
120 PRIN#2,A$;B$
█
```

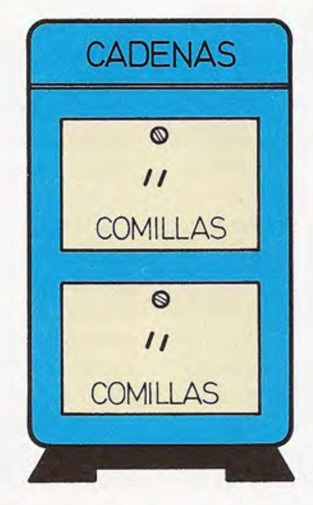
Esta secuencia de instrucciones almacenaría en el archivo número 2 lo siguiente:

MARICARMEN

Si, posteriormente, se extraen los datos de ese mismo archivo, ambos datos se procesarían como si de uno solo se tratase. Para distinguirlos es necesario introducir un separador entre ellos.

Empleando un comando PRINT# por cada dato se incluye automáticamente el carácter de retorno de carro como separador. Incorporando tal propiedad al último ejemplo, éste quedaría como sigue:



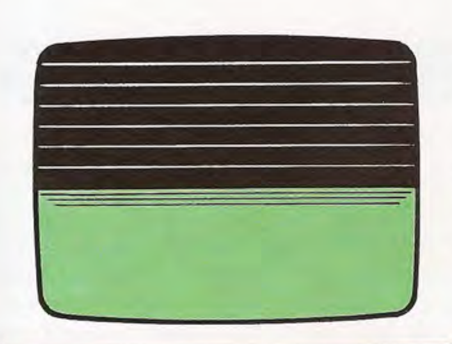







**SEPARADORES PARA ARCHIVOS**

*A la hora de almacenar datos en un archivo, entran en juego varios tipos de separadores. Entre ellos están el retorno de carro y la coma. En el caso de trabajar con datos numéricos, se considera al espacio en blanco como un separador más, mientras que al operar con cadenas hay que considerar el uso de las comillas.*



OPEN "0",# 1,"SCRN:"

*Los comandos para tratamiento de archivos secuenciales permiten también la comunicación con otros periféricos, por ejemplo, con la pantalla u órgano de visualización.*

```
100 A$="MARI"
110 B$="CARMEN"
120 PRINT#2,A$
130 PRINT#2,B$
█
```

Cuyo resultado en el archivo mostraría el siguiente aspecto:

MARI§CARMEN

Con el símbolo § se ha querido representar el carácter de retorno de carro.

Así pues, el primer delimitador o separador de datos es el carácter de retorno de carro. Este, como se ha visto, se introduce automáticamente al final de cada lista de datos.

En todo caso, el carácter de retorno de carro no es el único separador permitido. También puede utilizarse como separador el carácter «coma». De esta forma será posible incluir varios datos en el argumento de PRINT#:

```
100 A$="MARI"
110 B$="CARMEN"
120 PRINT# 2,A$;",";B$
```

En esta ocasión, entre uno y otro dato se almacenará el carácter coma. En el archivo ello quedaría como sigue:

MARI,CARMEN

Al ser la coma un separador válido, ambos datos estarán perfectamente delimitados. En consecuencia, la posterior ejecución de la instrucción:

```
200 INPUT# 2,T$,P$
```

asignaría a las variables T$ y P$ los valores MARI y CARMEN respectivamente. Pero esto no es todo. Cualquier coma será leída como delimitador de datos. Incluso las comas que hayan sido introducidas como parte de un dato. Así, el dato alfanumérico "PEREZ,JOSE" será leído de un archivo como dos cadenas distintas: "PEREZ" y "JOSE".

Para que sea posible incluir el carácter coma en el interior de una cadena alfanumérica se hace necesario delimitar ésta de otro modo. Puede recurrirse a un nuevo carácter separador: las comillas.

Al ejecutar un comando INPUT#, si éste encuentra como primer carácter a las comillas, tomará como límite del dato las siguientes comillas. Supóngase, por ejemplo, que en el archivo se encuentra almacenada la siguiente serie de caracteres:

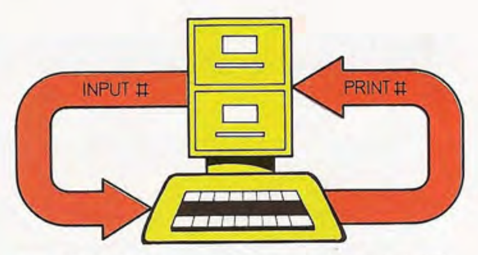
"PEREZ,JOSE"

Si se ejecuta, a continuación, el comando INPUT#1,T$, la variable alfanumérica T$ adoptará el valor PEREZ,JOSE.

Resumiendo, existen tres caracteres que sirven como delimitadores de datos alfanuméricos en el seno de un archivo secuencial: el retorno de carro, la coma y las comillas. En el caso de datos numéricos se emplea un cuarto delimitador, el espacio en blanco.

## Empleo de los delimitadores

De los delimitadores anteriormente citados tan sólo dos se almacenan automáticamente en el archivo: el retorno de carro, al final de una lista de datos,




*El diálogo entre la unidad central del ordenador y los archivos de información se materializa mediante los comandos PRINT# e INPUT#, según se ordene una operación de escritura o lectura en el archivo.*

y el espacio en blanco, entre datos numéricos de poca longitud.

Se ha visto ya la forma de intercalar un carácter coma. Para la grabación del carácter comillas la cosa se complica un poco más. Lo mejor en este caso es utilizar el código ASCII del referido carácter. Con ello, la grabación del dato asociado al último ejemplo se efectuaría con la siguiente línea de programa:

```
350 PRINT#1,CHR$(34);"PEREZ,JO
    SE";CHR$(34)...
```

En efecto, se ha utilizado la función CHR$ para obtener el carácter comillas, carácter cuyo código ASCII suele coincidir con el número 34.

Esta es una forma de intercalar delimitadores entre los datos a almacenar, aunque, no cabe duda que el método resulta ligeramente engorroso.

En algunos dialectos BASIC se dispone de un comando que inserta automáticamente los delimitadores adecuados. El mencionado comando se asocia a la palabra WRITE#, cuya formulación es similar a la del comando PRINT#.

(Número de línea) WRITE# <número>, <dato1>[,<dato2>...]

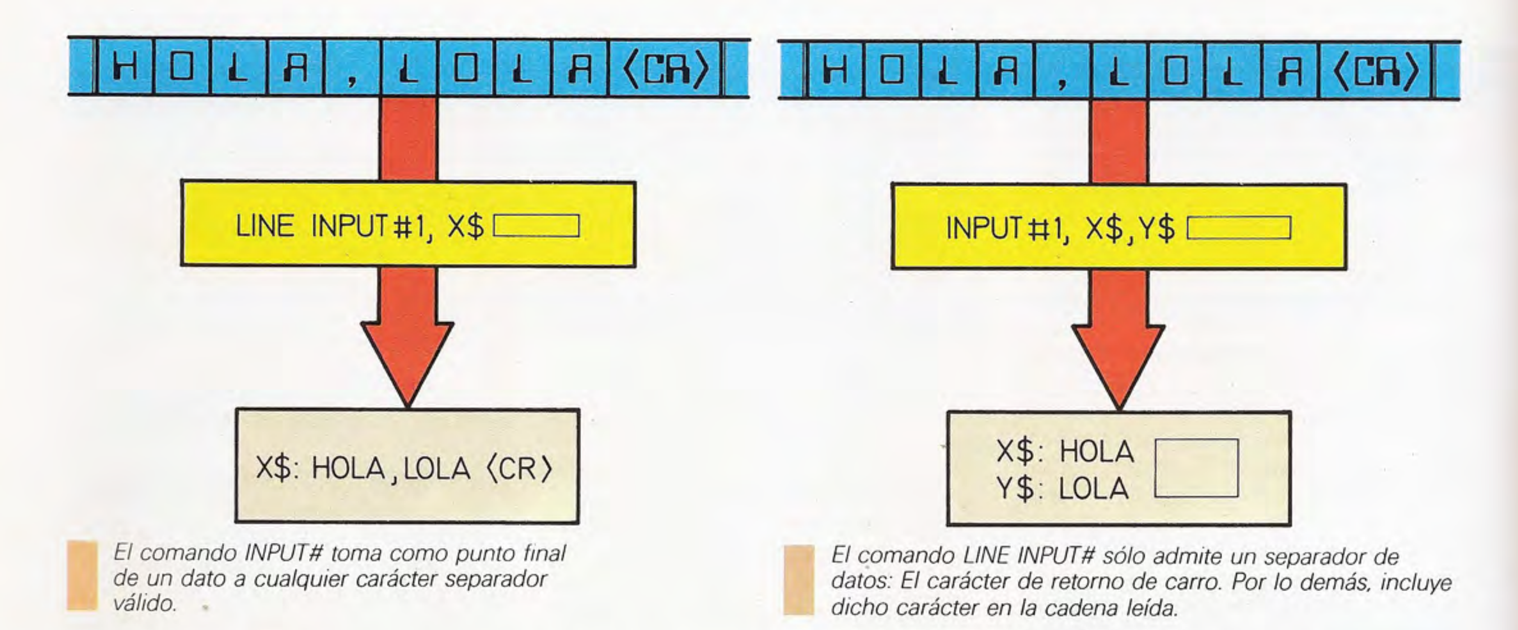
El comando WRITE# inserta comas

### EOF

Adopta el valor lógico CIERTO cuando se alcanza el final de un archivo.

*Formato:* EOF (<número de archivo>)

*Ejemplos:* 20 IF EOF(1) THEN END




*El comando INPUT# toma como punto final de un dato a cualquier carácter separador válido.*

*El comando LINE INPUT# sólo admite un separador de datos: El carácter de retorno de carro. Por lo demás, incluye dicho carácter en la cadena leída.*

entre los datos que se van escribiendo en el archivo. Los datos de tipo alfanumérico se graban encerrados entre comillas. Además, WRITE# proporciona un carácter de retorno de carro al final de cada lista de datos. En definitiva, este comando deja los datos perfectamente delimitados en el archivo.

Al igual que PRINT#,WRITE# tiene un comando homólogo para la representación en pantalla. Se trata de WRITE (sin el signo # al final). WRITE actúa en pantalla de la misma forma que WRITE# en un archivo. Véase un ejemplo:

```
PRINT "DOS Y DOS SON ",
      2+2 DOS Y DOS SON 4
WRITE "DOS Y DOS SON ",
      2+2 "DOS Y DOS SON ",4
█
```

En él se pone de manifiesto la clara diferencia que existe entre PRINT y WRITE.

Para la escritura en un archivo secuencial también se puede hacer uso de PRINT# con la opción USING. Esta opción permite definir el formato con el que van a ser grabados los datos. Su funcionamiento es totalmente análogo al de PRINT USING, al que ya se ha dedicado un capítulo en esta obra.

Por lo que respecta a la lectura de los datos contenidos en el archivo, es preciso indicar el modo en el que ésta se efectúa. El comando utilizado hasta ahora se formula a partir de la palabra clave INPUT#. Ya se ha indicado que este comando distingue varios delimitadores de datos:

- Datos numéricos:
  Espacio en blanco
  Coma
  Retorno de carro
- Datos alfanuméricos:
  Coma
  Retorno de carro
  Comillas

En el caso de las comillas, se toma como un solo dato todo lo encerrado entre dos de esos caracteres.

Estos caracteres son precisamente los que emplea el comando INPUT# para identificar el final de un dato. De esta forma, el binomio WRITE#/INPUT# permite un fácil y eficiente manejo de los archivos secuenciales.

El BASIC suele ofrecer aún otro comando dedicado a la recuperación de datos de un archivo secuencial; comando que admite como único delimitador el carácter de retorno de carro. Para su puesta en práctica se emplean las palabras LINE INPUT#, de acuerdo al formato que se expone a continuación.

(Número de línea) LINE INPUT#
<número>,<variable cadena>

La ejecución de este comando asigna un valor a la variable de cadena especificada; valor que corresponderá a la serie de caracteres que se lean del archivo. En dicha serie de caracteres se toma el retorno de carro como único delimitador. Es más, el propio carácter de retorno de carro se considerará como parte del dato.

El comando LINE INPUT# deriva del comando análogo reservado para la lec-

**WRITE#**

Escribe datos en un archivo introduciendo los separadores adecuados entre ellos.

*Formato:* WRITE#<número>,<dato>[;<dato>...]

*Ejemplos:* 20 WRITE #1,"PERICO"
70 WRITE #2, A+B

tura de datos introducidos mediante el teclado: LINE INPUT (como siempre, sin #).

Este último realiza la misma acción que el comentado LINE INPUT#. La única diferencia entre ambos reside en el dispositivo del que se toman los datos. Así pues, LINE INPUT es equivalente al comando INPUT, pero con la salvedad de que sólo admite variables de cadena y que toma el retorno de carro como un carácter más de la cadena (el último de la misma).

## WRITE

Escribe datos en pantalla separándolos por comas; las cadenas se representan entre comillas.

*Formato:* WRITE <dato1> [,<dato2>...]

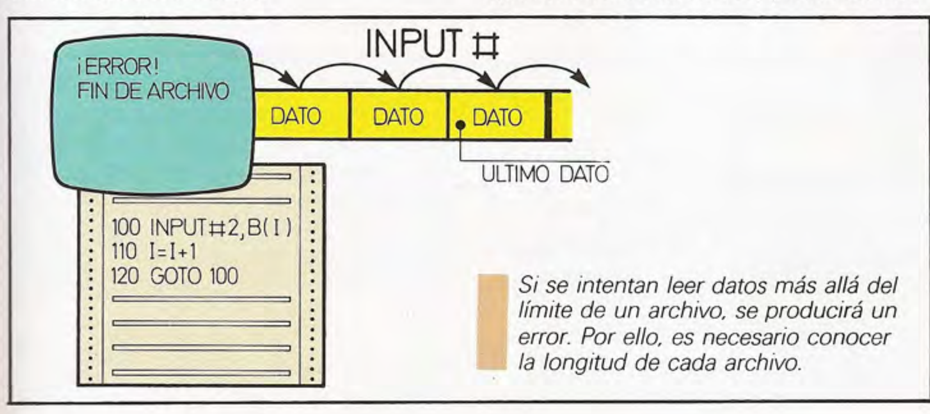
*Ejemplos:*
```
20 WRITE B,A$
70 WRITE "A=",352
```

## Archivos y dispositivos asociados

Todo lo concerniente a los archivos secuenciales se asocia, por regla general, al dispositivo de almacenamiento más habitual: el casete. Sin embargo, se pueden crear archivos secuenciales en otros periféricos.

Los dos soportes más empleados para el almacenamiento de datos son las cintas de casete y los discos magnéticos. Al respecto, cabe añadir que el uso de archivos secuenciales almacenados en disco es idéntico al de los archivos en casete.

Aparte de los archivos de almacenamiento de datos existe otro tipo de archivos. Se trata de los archivos abiertos en otros dispositivos que no son propiamente de almacenamiento. En realidad no se trata de archivos en el sentido estricto de la palabra; sino que más bien habría que definirlos como canales de comunicación con periféricos. Para la manipulación de estos canales se emplean los mismos comandos que para el manejo de archivos secuenciales. Es por ello por lo que, a menudo, se identifican con los archivos reales.




*Si se intentan leer datos más allá del límite de un archivo, se producirá un error. Por ello, es necesario conocer la longitud de cada archivo.*

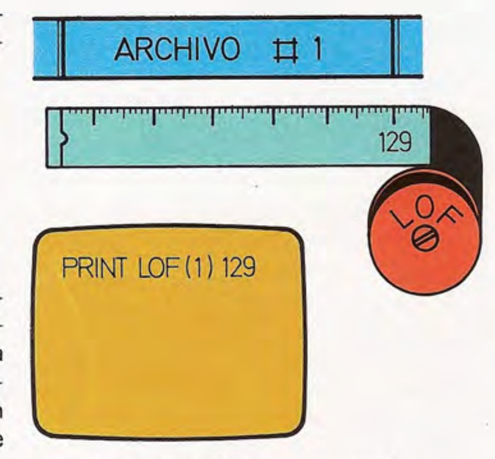
Se pueden abrir canales para la comunicación con los siguientes periféricos:

— Magnetófono de casetes.
— Unidad de disco.
— Pantalla.
— Impresora.
— Modem o red local.

En la lista se han incluido las unidades de almacenamiento externo (magnetófono y unidad de disco), ya que la apertura de un archivo en ellas lleva implícita la comunicación a través de un canal. En el último lugar de la lista se ha mencionado dos modalidades de comunicación genérica.

El modem es un dispositivo que permite la comunicación entre ordenadores distantes, haciendo uso de la red telefónica. Las redes locales son agrupaciones de ordenadores conectados entre sí. Por regla general, éstos comparten ciertos periféricos como puede ser la impresora. Como su nombre indica, una red local une ordenadores (y periféricos) que se encuentran próximos (a menudo en la misma habitación o habitaciones contiguas).

La indicación del dispositivo adecuado se suele incluir en el comando de apertura del canal OPEN, y más concretamente en el nombre del archivo. Así, para la apertura de un canal asociado al archivo DATOS, se pueden especificar los siguientes nombres en el argumento de OPEN.




*La función LOF proporciona la longitud, medida en bytes, de un determinado archivo.*

"CAS:DATOS"
"D:DATOS"
"SCRN:DATOS"
"KYBD:DATOS"
"LPT:DATOS"

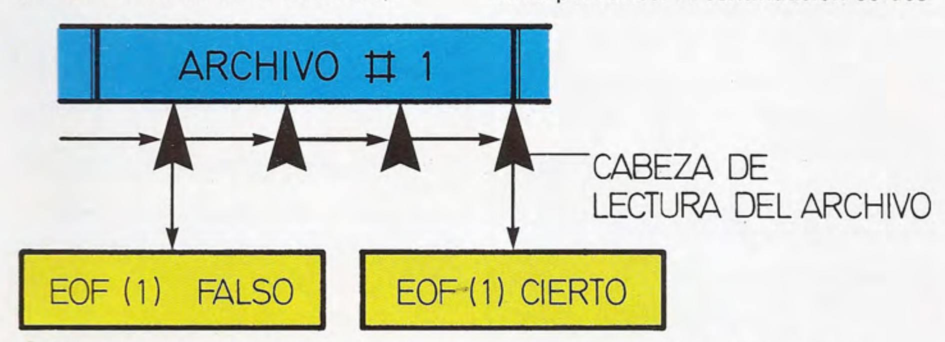
Los caracteres que preceden a los dos puntos especifican el periférico. En el ejemplo, corresponden a: casete, unidad de disco, pantalla, teclado e impresora. Estas formulaciones no están estandarizadas. Aquí se ha tomado una muy semejante a la adoptada por el BASIC de Microsoft.

La identificación de periféricos se realiza de formas muy heterogéneas, dependiendo de cada aparato en particular.

En algunos casos, cada dispositivo tiene asignado un número y se consideran permanentemente abiertos. Así, basta-

| TABLA DE CONVERSIÓN (1) | | | | | | |
|---|---|---|---|---|---|---|
| Ordenador | Delimitación del archivo | | Escritura | | Lectura | |
| | LOF (<n>) | EOF (<n>) | WRITE # | WRITE | LINE INPUT | LINE INPUT # |
| AMSTRAD | — | EOF | WRITE #9 | — | — | LINE INPUT #9 |
| APPLE II (APPLESOFT) | — | — | — | — | — | — |
| APRICOT (M-BASIC) | LOF (<n>) | EOF (<n>) | WRITE # | WRITE | LINE INPUT | LINE INPUT # |
| ATARI | — | — | — | — | — | — |
| CBM 64 | — | — | — | — | — | — |
| DRAGON | — | EOF (<n>) | — | — | LINE INPUT | — |
| EQUIPOS MSX | LOF (<n>) | EOF (<n>) | — | — | LINE INPUT | LINE INPUT # |
| HP-150 | LOF (<n>) | EOF (<n>) | WRITE # | WRITE | LINE INPUT | LINE INPUT # |
| IBM PC | LOF (<n>) | EOF (<n>) | WRITE # | WRITE | LINE INPUT | LINE INPUT # |
| MPF | — | — | — | — | — | — |
| NCR DM-V (MS-BASIC) | LOF (<n>) | EOF (<n>) | WRITE # | WRITE | LINE INPUT | LINE INPUT # |
| NEW BRAIN | — | — | — | — | LINPUT | LINPUT # |
| ORIC | — | — | — | — | — | — |
| SHARP MZ-700 (MZ-BASIC) | — | — | — | — | — | — |
| SINCLAIR QL | — | EOF (<n>) | — | — | — | — |
| SPECTRAVIDEO | LOF (<n>) | EOF (<n>) | — | — | LINE INPUT | LINE INPUT # |
| ZX-SPECTRUM | — | — | — | — | — | — |

ría con teclear PRINT#3,A para mandar el dato A a la impresora (suponiendo que el 3 fuera su canal asociado).




*Para localizar el límite de un archivo se emplea la función EOF. Dicha función lógica adopto el valor CIERTO cuando se alcanza el mencionado límite.*

Abriendo los canales adecuados se puede establecer la comunicación con cada periférico. A continuación se describe el posible empleo que se puede hacer de cada uno.

- Teclado:

El teclado es un dispositivo de entrada de datos. Ello significa que sólo se pueden recibir datos y nunca mandarlos hacia el teclado. En la mayor parte de los casos no es necesario abrir un canal para la comunicación con este periférico. De utilizar un canal, el resultado de ejecutar los comandos INPUT# y LINE INPUT# será idéntico al de INPUT y LINE INPUT.

- Pantalla:

El caso de la pantalla es similar al del

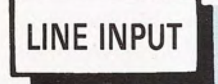
| TABLA DE CONVERSIÓN (2) | | | | | | |
|---|---|---|---|---|---|---|
| Ordenador | Caracteres identificadores de periféricos | | | | | |
| | Casete | Disco | Impresora | Teclado | Pantalla | Comunicaciones |
| AMSTRAD | #9 | #9 | #8 | — | #1 ... #7 | — |
| APPLE II (APPLESOFT) | — | — | — | — | — | — |
| APRICOT (M-BASIC) | — | — | — | — | — | — |
| ATARI | "C: | "D: | "P: | "K: | "S: | "R: |
| CBM 64[1] | 1 | 8 | 4 | — | Ø | — |
| DRAGON[2] | -1 | — | -2 | — | — | — |
| EQUIPOS MSX | "CAS: | — | "LPT: | — | "CRT: "GRP:[3] | — |
| HP-150 | — | — | — | — | — | — |
| IBM PC | "CAS1: | "A:<br>"B: | "LPT:<br>"LPT2:<br>"LPT3: | "KYBD: | "SCRN: | "COM1:<br>"COM2: |
| MPF | — | — | — | — | — | — |
| NCR DM-V (MS-BASIC) | — | — | — | — | — | — |
| NEW BRAIN[4] | 1 - 2 | — | 8 | 5 | Ø | 9 |
| ORIC | — | — | — | — | — | — |
| SHARP MZ-700 (MZ-BASIC) | — | — | — | — | — | — |
| SINCLAIR QL | — | MDV[5] | — | CON | SCR | SER |
| SPECTRAVIDEO | "CAS: | 1 - 2 | "LPT: | "KYBD: | "SCRN: | — |
| ZX-SPECTRUM | — | — | — | — | — | — |

[1] El número indicado se ha de adjuntar en el segundo parámetro del comando OPEN. [2] El identificador se sitúa en la posición del número de archivo, dentro del comando OPEN. [3] CRT abre el canal en la pantalla de texto, GRP lo hace en la pantalla de gráficos. [4] El número indicado se ha de colocar como segundo parámetro en el comando OPEN. [5] En este caso el dispositivo no es el disco sino el Microdrive de Sinclair.

teclado, con la única diferencia de que ésta sólo admite comandos de salida de datos. La apertura de un canal para la pantalla suele ponerse en práctica en los ordenadores que disponen de varios modos gráficos. En tal caso se utiliza el canal para escribir caracteres cuando se está en un modo gráfico que no admite texto directamente.

- Impresora:

  Determinados ordenadores no poseen instrucciones BASIC apropiadas para el manejo de la impresora. Cuando sucede esto, el usuario se ve obligado a crear un canal y trabajar con la impresora como si se trata de un archivo de datos. De nuevo, los únicos comandos utiliza-




Recoge una cadena del teclado incluyendo en ella el carácter de retorno de carro.

*Formato:* LINE INPUT <var1>[,<var2>...]

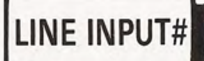
*Ejemplos:* 20 LINE INPUT CA$

bles son los de salida de datos: PRINT# y WRITE#.

Normalmente, en la apertura de canales para estos periféricos no es necesario adjuntar un nombre de archivo. Para crear un canal para la comunicación con la impresora basta con ejecutar la orden:

OPEN "O",#1,"LPT:"

En efecto, no es necesario identificar un archivo específico ya que, en realidad, tal archivo es puramente ficticio.



## LINE INPUT#

Lee datos de un archivo tomando como único delimitador el carácter de retorno de carro.

*Formato:* LINE INPUT# <número>,<var1>[,<var2>...]

*Ejemplos:*
```
10 LINE INPUT#1,A$
50 LINE INPUT#2, R$,P$,S$
```

```
A$ = "MARI"
B$ = "CARMEN"
PRINT#2, A$; B$
```

| M | A | R | I | C | A | R | M | E | N | (CR) |

*Si en el argumento de PRINT# se separan los datos con punto y coma, éstos se escribirán de forma contigua al efectuar dicha operación.*

```
A$ = "MARI"
B$ = "CARMEN"
PRINT#2, A$
PRINT#2, B$
```

| M | A | R | I | (CR) | C | A | R | M | E | N | (CR) |

*El comando PRINT# introduce al final de cada lista de datos un carácter de retorno de carro. Tal carácter permite distinguir unos datos de otros.*

```
A$ = "MARI"
B$ = "CARMEN"
PRINT#2, A$; ","; B$
```

| M | A | R | I | , | C | A | R | M | E | N | (CR) |

*Una forma de separar datos en un archivo consiste en introducir el signo «coma» entre cada dos de ellos.*

```
A$= "MARI"
B$ = "CARMEN"
PRINT#2, CHR$(34);A$; CHR$(34)
```

| " | M | A | R | I | " | (CR) |

*En el caso de las cadenas de caracteres, éstas se pueden delimitar introduciendo el signo de comillas antes y después de cada dato.*

```
A$ = "MARI"
B$ = "CARMEN"
WRITE#2, A$, B$
```

| " | M | A | R | I | " | , | " | C | A | R | M | E | N | " | (CR) |

*El empleo del comando WRITE evita la necesidad de tener que introducir separadores de datos. El propio comando se encarga de colocar los separadores adecuados.*

```
A = 12
B = 57
PRINT#2,USING "###"; A; B
```

| X | 1 | 2 | X | 5 | 7 | (CR) |

ESPACIOS EN BLANCO

*Otra forma de separar correctamente los datos consiste en especificar su formato mediante la opción USING del comando PRINT#.*

# Archivos aleatorios (I)

## Creación y uso de archivos de acceso directo

Los elementos constitutivos de un archivo secuencial (los registros) se graban uno detrás de otro, y para acceder a uno de ellos es preciso pasar por todos los anteriores. Evidentemente, esto plantea algunas dificultades a la hora de manejar la información contenida en ese tipo de archivos. Así pues, en muchos casos es necesario recurrir a los archivos de acceso directo que obvian algunas de estas dificultades. Dos de estas características ventajosas de los archivos de acceso directo son las siguientes:

- En primer lugar, en un archivo directo no es necesario grabar los registros uno tras otro, sino que se pueden grabar en la posición y en el orden que se desee.
- En segundo lugar, a la hora de acceder a uno de los elementos del archivo, no es necesario acceder previamente a todos los demás.

Un archivo en general es un conjunto de registros. En el caso de los archivos aleatorios, cada registro está formado por una serie de campos que también han de tener una longitud fija, de manera que el registro tendrá una estructura fija. Esta estructura del registro ha de ser elegida a la vista de la información que se va a grabar en el archivo. Dentro de estos campos, existe uno muy importante: la clave, que decide la posición relativa de cada registro dentro del archivo. Por lo general, la clave suele ser numérica (en el caso del BASIC siempre lo será). La clave puede equipararse a un índice que permite el acceso a un determinado registro.

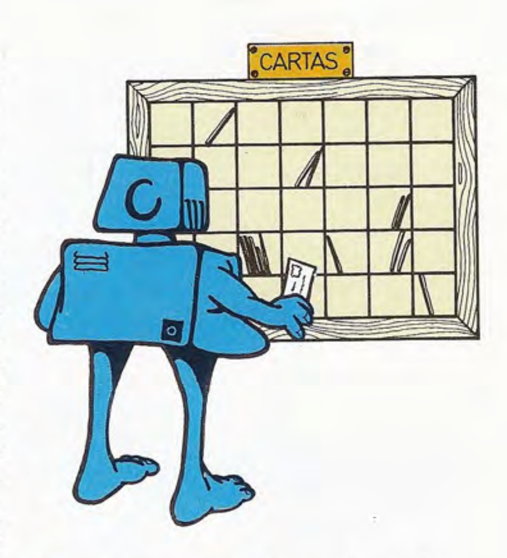
El primer paso para construir un archivo de acceso directo es crear la estructura de los registros que contendrán la información. Así, si se desea crear un archivo con los números de teléfono y las direcciones de los amigos, será necesario que cada registro contenga, por ejemplo, las siguientes informaciones:

Nombre
Primer apellido
Segundo apellido
Dirección
Ciudad
Teléfono




*El casete es un soporte de almacenamiento externo capaz exclusivamente de albergar archivos de tipo secuencial.*

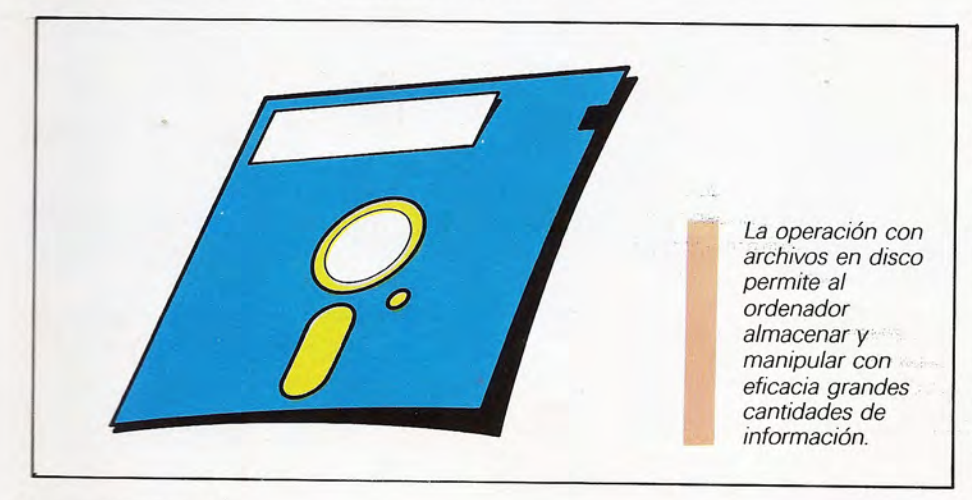
La longitud reservada para cada uno de estos campos ha der ser definida y permanecerá fija a lo largo de todo el proceso. Esta longitud se elegirá de forma que quepa en el campo la información que se desea almacenar. Por ejemplo, para almacenar el campo nombre será suficiente con reservar 20 caracteres; la mayor parte de los nombres cabrán en ese sitio. Desde luego, en este punto puede ser preciso llegar a una solicitud de compromiso; no todos los nombres caben en 20 caracteres, pero si reservamos mayor espacio nos encontraremos con que la mayor parte de los nombres no llegarán a ocupar gran parte del espacio disponible y, por tanto, ese espacio se perderá. Hay que tener en cuenta que si reservamos mucho espacio para cada uno de los campos, el registro en su conjunto resultará más voluminoso y, en consecuencia, cabrán menos registros en el soporte de memoria sobre el que reside el archivo. En el




*La diferencia fundamental entre los archivos de tipo secuencial y de acceso directo reside en que, en estos últimos, es posible acceder a cualquiera de sus elementos sin pasar por los anteriores.*




*Los archivos de acceso directo exigen soportes de memoria que permitan un acceso aleatorio a la información almacenada; tal es el caso de las unidades de disco flexible.*



*La operación con archivos en disco permite al ordenador almacenar y manipular con eficacia grandes cantidades de información.*

## OPEN

Realiza la apertura de un archivo en un dispositivo de almacenamiento externo.

*Formato:* OPEN "R",#<n1>,<nom>,<long>

*Ejemplos:*

```
10 OPEN "R,#1,"DATOS",123
50 OPEN "R",#2,"ARCHIVO.DAT",50
```

## FIELD

Especifica el formato del registro

*Formato:* FIELD#<n1>,<long1> AS <var1>,...

*Ejemplos:*

```
20 FIELD#1, 20 AS A$, 10 AS B$
70 FIELD#2, 5 AS TEL$, 15 AS NOM$
```

ejemplo propuesto, una distribución más o menos racional de las longitudes de los registros puede ser la siguiente:

| | |
|---|---|
| Nombre | (20 caracteres) |
| Primer apellido | (15 caracteres) |
| Segundo apellido | (15 caracteres) |
| Dirección | (30 caracteres) |
| Ciudad | (10 caracteres) |
| Teléfono | (9 caracteres) |

## Operando desde el BASIC

Las operaciones que es necesario efectuar para crear uno de estos archivos desde el BASIC son las siguientes:

1) Abrir el archivo en modalidad de acceso directo. A diferencia de lo que sucedería con los archivos secuenciales, aquí no es preciso abrir el archivo en modo de lectura o de escritura. Una vez abierto el archivo, en él se pueden efectuar operaciones tanto de lectura como de escritura.

La siguiente instrucción opera abriendo el archivo en caso de que exista; y en caso de que no exista, lo crea:

```
OPEN "R",#n1,"<nombre>",n2
```

Al comando OPEN le siguen una serie de parámetros que es necesario especificar. En primer lugar, la R entre comillas indica que se trata de un archivo de acceso directo. A continuación, n1 indica el número de canal que se desea abrir. El nombre que se especifique a continuación es el del archivo con el que se desea trabajar. Por último, n2 indica la longitud total del archivo.

Existen limitaciones en cuanto al tamaño máximo del registro, pero éstas dependen del sistema con el que se esté trabajando.

2) Especificar la estructura de cada registro (esta operación no es necesaria en todas las versiones del BASIC).

```
FIELD # n1, n3 AS<var1>, n4 AS <var2>,...
```

Mediante esta instrucción se define el campo asociado a un canal determinado, que se identifica mediante el número n1. A continuación se indica la estructura mediante la enumeración de

las variables asociadas a cada campo: var1, var2, etc., siendo precedidas estas variables por un número que indica el espacio que se les desea reservar.

Un detalle importante a tener en cuenta es que en un archivo secuencial todos los campos que componen el registro han de ser alfanuméricos. El hecho de que estos campos hayan de ser obligatoriamente alfanuméricos no quiere decir que en un archivo no se pueden grabar números, sino que sobre ellos habrá que realizar las transformaciones adecuadas para convertirlos en cadenas de caracteres.

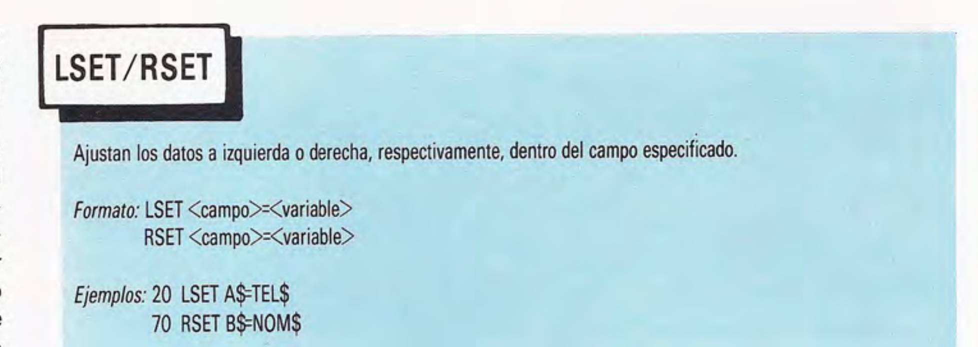
3) Por lo general, los datos que se van a introducir en el archivo no tienen la misma longitud que los campos que les fueron reservados. Esta es la razón de que sea preciso realizar una adaptación de los datos a los campos donde se alojarán dentro del archivo. Este ajuste puede realizarse mediante dos instrucciones existentes a tal efecto. Estas instrucciones se encargan de ajustar los caracteres bien a la derecha o bien a la izquierda del campo disponible, y rellenan el resto del campo con espacios en blanco. Por lo general, se suele emplear la instrucción LSET con cadenas de caracteres alfanuméricos, mientras que RSET se suele emplear para cadenas de caracteres numéricos.

La transformación de un número en una cadena de caracteres es fácil de realizar. De hecho, ya se conoce una función capaz de realizar esa operación: se trata de la función STR$. Sin embargo, esta función, que en principio funciona correctamente, puede ser mejorada. Hay que observar que un número entero se almacena en la memoria del ordenador ocupando dos bytes, mientras que el campo preciso para almacenar ese mismo número en un archivo mediante la función STR$ debería tener 5 caracteres.

Para evitar este desperdicio del espacio de almacenamiento de los soportes externos, dispone de una función que permite almacenar un número de simple precisión en una cadena, empleando para ello tan solo dos bytes. El sistema seguido es el mismo que se emplea para almacenar ese número en memoria. La función a utilizar es MKI$.

Existen también funciones equivalentes para almacenar números de simple precisión en memoria mediante cuatro bytes (MKS$) y números de doble precisión mediante ocho bytes (MKD$).

Por supuesto, existen funciones para realizar la operación inversa, funciones que obtienen el número primitivo a partir de la cadena generada por MKI$, MKS$ y MKD$. Estas funciones serán estudiadas más adelante al explicar como se puede leer un archivo.

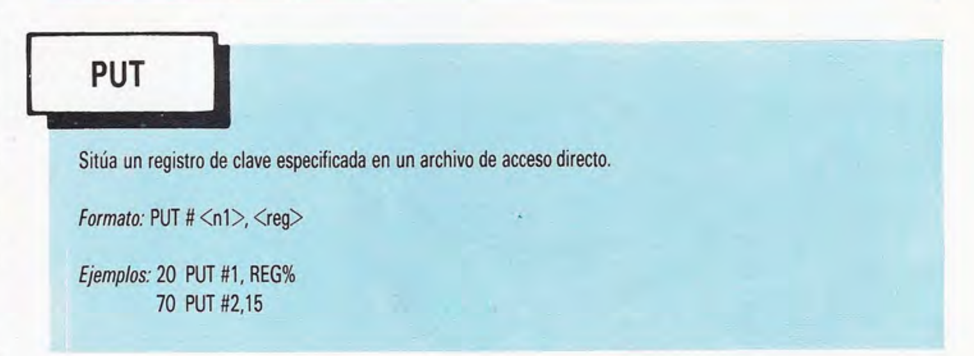


## LSET/RSET

Ajustan los datos a izquierda o derecha, respectivamente, dentro del campo especificado.

*Formato:* LSET <campo>=<variable>
RSET <campo>=<variable>

*Ejemplos:* 20 LSET A$=TEL$
70 RSET B$=NOM$



## PUT

Sitúa un registro de clave especificada en un archivo de acceso directo.

*Formato:* PUT # <n1>, <reg>

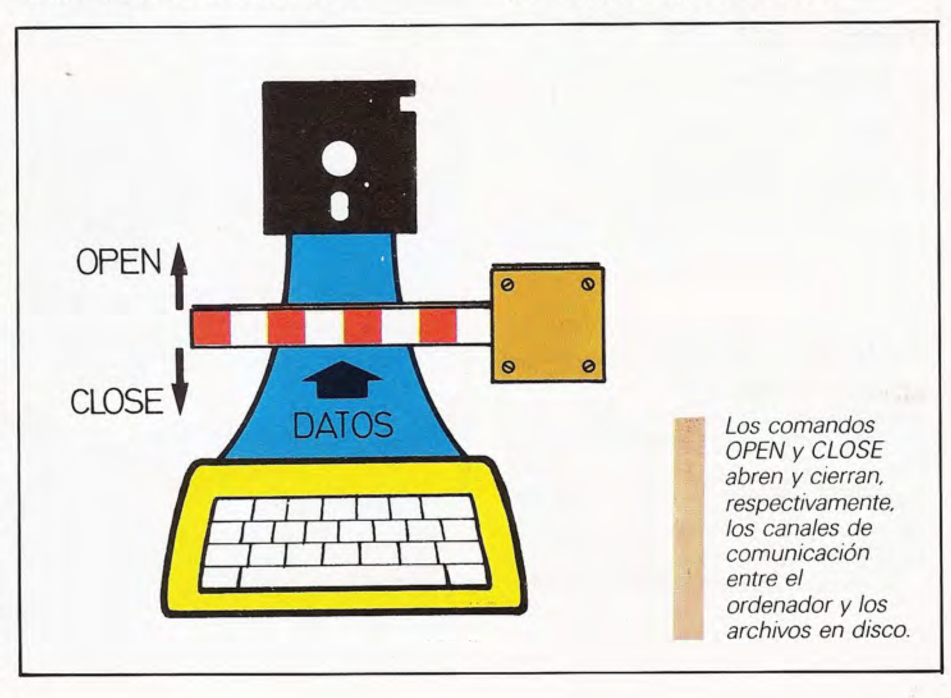
*Ejemplos:* 20 PUT #1, REG%
70 PUT #2,15

La formulación de LSET es la que sigue:

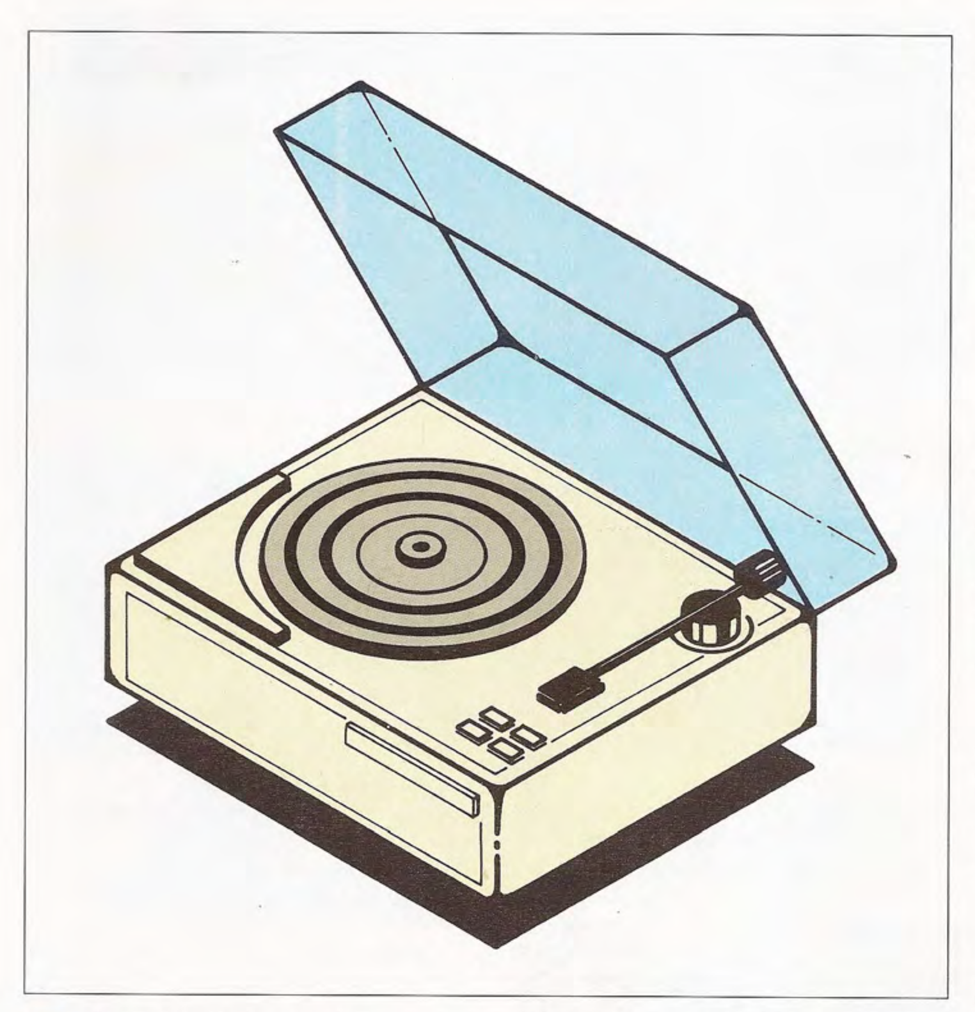
LSET <var. dest.>=<var. origen>

En donde:

<var. dest.>: Variable donde se desea que la cadena quede ajustada.




*Los comandos OPEN y CLOSE abren y cierran, respectivamente, los canales de comunicación entre el ordenador y los archivos en disco.*



*Al igual que ocurre con la reproducción de un disco musical, los archivos de acceso directo permiten el libre acceso a cualquiera de sus registros. Tomando la analogía del disco, el acceso a un registro equivale a posicionar la cabeza en cualquier parte de la superficie del mismo.*

<var. origen>: Variable donde se en cuentra la cadena original.

4) Introducir los datos en el buffer, de donde pasarán en el momento oportuno al archivo en disco.

La posición relativa dentro del archivo queda definida por POS%, que es la clave del registro y que en este caso es numérica. Esta clave ha de ser un número entero. Si existen más datos a introducir, se ha de volver al punto 3; si no, se pasa al siguiente punto.

```
PUT # n1,POS%
```

5) Cerrar el archivo que se ha empleado. Esta operación es muy importante ya que los elementos del archivo se van grabando en el buffer y van pasando al disco en el momento oportuno. La operación de cierre del archivo consiste en disociar el canal del archivo, transfiriendo antes el contenido del buffer al archivo.

```
CLOSE # 1 ó END
```

## Creación de un archivo

Un ejemplo de creación de un archivo de este tipo puede ser el que se detalla en los próximos párrafos.

La primera operación se realiza en la línea 1000. Concretamente, en este caso se abre el canal 1; el nombre del archivo será DATOS y la longitud reservada para cada registro es de 98 bytes:

```
1000 OPEN "R", # n1, "DATOS", 98
```

A continuación, es necesario especificar la estructura de los registros que se van a emplear.

```
1010 FIELD # N1, 20 AS N OM$, 15 AS AP1$, 15 AS AP2$,
30 AS DIR$, 10 AS CIU$#, 9 AS TEL$
```

Con la línea 1020 se pretende leer el número de registro en el que será grabada la información que se introducirá más adelante:

```
1020 INPUT "QUE REGISTRO ";REG%: IF REG%=0 THEN
     CLOSE # N1: END
```

### GET

Lee el registro especificado de un archivo de acceso directo.

*Formato:* GET #<n1>,<reg>

*Ejemplos:*
```
10 GET #1, REG%
50 GET #2,25
```

### CVI

Convierte una cadena que representa un número en formato comprimido en uno de precisión entera.

*Formato:* CVI (<cadena>)

*Ejemplos:* 20 N=CVI(N$)

## CVD

Convierte una cadena que representa un número en formato comprimido en uno de doble precisión.

*Formato:* CVD(<cadena>)

*Ejemplos:* 20 P=CVD(LL$)

## CVS

Convierte una cadena que representa un número en formato comprimido en uno de simple precisión.

*Formato:* CVS(<cadena>)

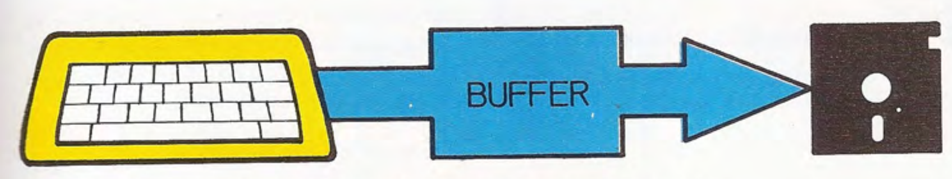
*Ejemplos:* 20 KG=CVS(MIL$)

## MKI$

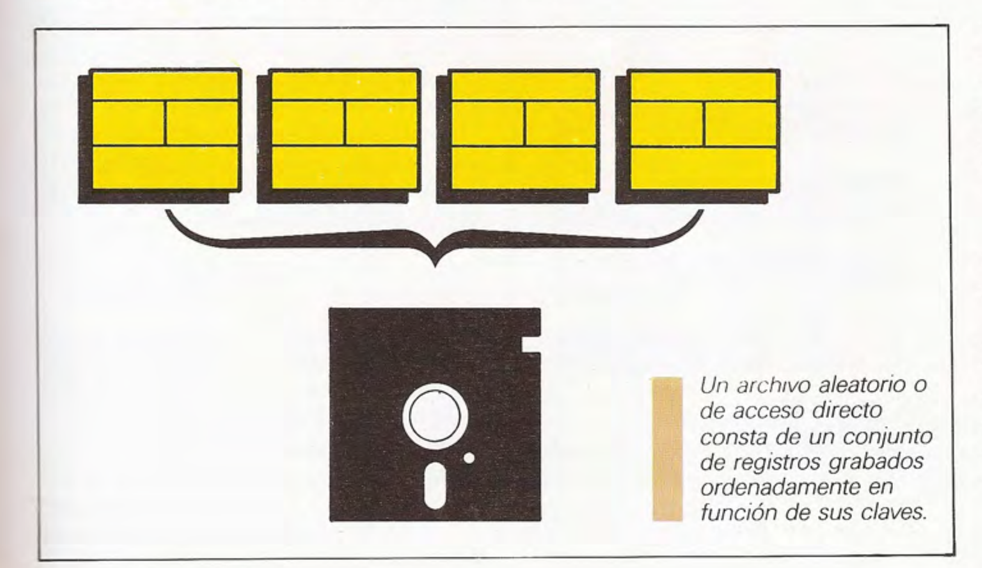
Convierte un número de precisión entera en una cadena en formato comprimido.

*Formato:* MKI$(<num>)

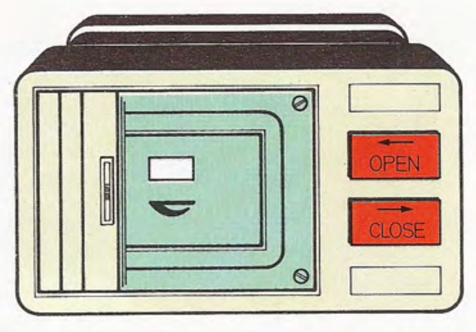
*Ejemplos:* 10 LET A$=MKI$(A)




*El «buffer» actúa como intermediario entre los datos generados por el proceso y el soporte donde se almacenan.*



*Un archivo aleatorio o de acceso directo consta de un conjunto de registros grabados ordenadamente en función de sus claves.*




*La apertura y cierre de un archivo creado en un dispositivo de almacenamiento externo corre a cargo, respectivamente, de las órdenes OPEN y CLOSE.*

Para salir del proceso basta con responder a la orden INPUT que el registro que se desea grabar es el registro 0; en tal caso se cerrará el archivo y se detendrá el proceso.

Evidentemente, antes de almacenar los datos en el archivo es preciso disponer de dichos datos. Por lo general, estos datos suelen proceder del exterior y son contados a través de la ejecución de instrucciones de tipo INPUT. También podrían derivar de procesos ejecutados con anterioridad, pero este caso es menos frecuente. Lo que sí sucede a veces es que la información recogida es tratada antes de ser almacenada en el archivo.

Aquí está, en definitiva, el bloque de instrucciones para la captación de los datos:

```
1030 INPUT "NOMBRE: ";N$
1040 INPUT "APELLIDO 1: ";A1$
1050 INPUT "APELLIDO 2: ";A2$
1060 INPUT "DIRECCION :";D$
1070 INPUT "CIUDAD :";C$
1080 INPUT "TELEFONO :";T#
```

A continuación, es preciso ajustar los valores que se desean grabar en el archivo al formato disponible:

```
1100 LSET NOM$=N$
1110 LSET AP1$=A1$
1120 LSET AP2$=A2$
1130 LSET DIR$=D$
1140 LSET CIU$=C$
1150 RSET TEL$=MKD$(T#)
```

Tras ello, puede ya grabarse la infor-

## MKS$

Convierte un número de precisión simple en una cadena en formato comprimido.

*Formato:* MKS$(<num>)

*Ejemplos:* 10 LET H$=MKS$(HS)

## MKD$

Convierte un número de precisión doble en una cadena en formato comprimido.

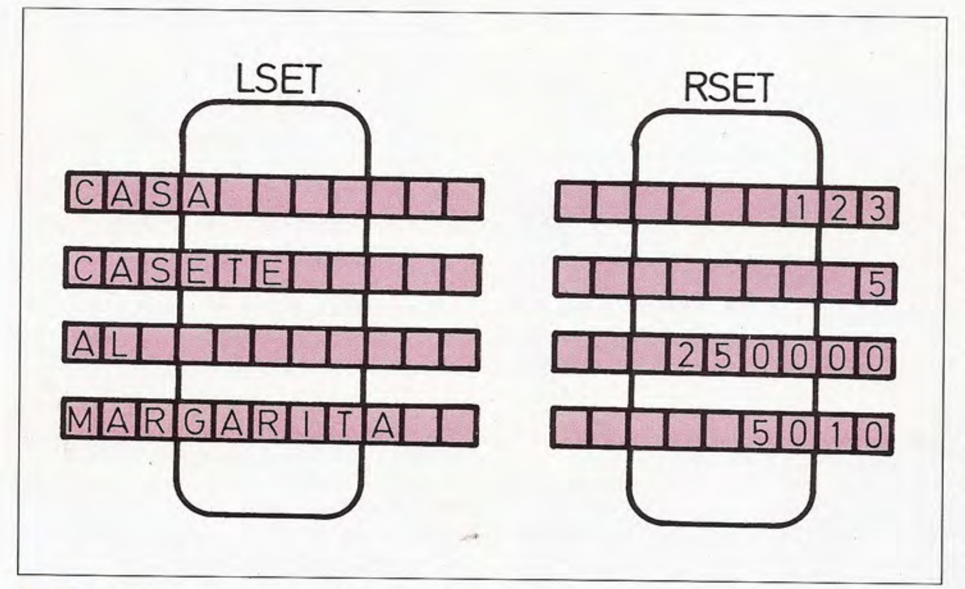
*Formato:* MKD$(<num>)

*Ejemplos:* 10 LET K$=MKD$(PESO)

NUM ...........
1.er APELLIDO ..................................
2.º APELLIDO ..................................
NOMBRE .........................................

CLAVE
CAMPO 1 | CAMPO 2
CAMPO 3

*Un registro es muy semejante a una ficha. Los espacios reservados en la ficha para los distintos conceptos reciben en los registros el nombre de «campos».*




*Mediante las funciones LSET y RSET, es posible ajustar los datos a izquierda o derecha, respectivamente, dentro de los campos del registro.*

mación relativa al registro que nos ocupa en el archivo:

```
1200 PUT #1,REG%
```

Por último, puede regresarse a la línea 1020 para tratar más entradas de datos:

```
1210 GOTO 1020
```

Junto al texto se reproduce el listado completo del programa ejemplo, destinado a la creación e introducción de datos en el archivo.

## Lectura de un archivo de acceso directo

Son cuatro los pasos a seguir para la lectura de los requisitos de un archivo de acceso directo o aleatorio; justamente, los que se detallan a continuación.

1) Abrir el archivo.

La apertura es análoga a la que se realiza al crear el archivo. Ante todo, al abrir el archivo para leer datos del mismo es preciso que este exista, esto es, que haya sido creado previamente. El nombre utilizado ha de ser el correcto y la longitud del registro ha de ser la misma con la que fue creado.

```
OPEN "R", # n1, "nombre", n2
```

2) Especificar la estructura de cada registro.

El registro ha de tener la misma estructura que tenía cuando el archivo fue creado. Lo único que ha de mantenerse es la estructura, ya que los datos están grabados en el archivo en registros de ese tipo. Sin embargo, los nombres de las variables asignadas a los campos del buffer pueden ser distintas:

```
FIELD# n1, n3 AS <var1>, n4 AS <var2>
```

3) Leer el registro deseado del archivo, con lo que dicha información quedará a la disposición del usuario para manejarlo según sus necesidades:

```
GEN# n1, REG%
```

Los datos alfanuméricos, como ya se ha dicho, están directamente a dispo-

| TABLA DE CONVERSIÓN | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Ordenador | OPEN | FIELD | Ajuste | Escritura/lectura | | Conversión | | Apertura |
| | OPEN "R",#n1, <nom>,n2 | FIELD#n1, n3AS<var1> | LSET/RSET | PUT#n1, <num> | GET#n1, <num> | CVI/CVS/ CVD | MKI$/MKS$/ MKD$ | OPEN<nom> [FOR<modo>] AS[#]n1 [LEN=n2] |
| AMSTRAD | — | — | — | — | — | — | — | — |
| APPLE II[1] (APPLESOFT) | OPEN<nom>, Ln1,Sn2,Dn2,Vn4 | — | — | WRITE nom, R1 | READ Nom, R1 | — | — | — |
| APRICOT (M-BASIC) | OPEN "R",#n1, <nom>,n2 | FIELD#n1, n3AS<var1>... | LSET/RSET | PUT#n1, <num> | GET#n1, <num> | CVI/CVS/ CVD | MKI$/MKS$/ MKD$ | OPEN<nom> [FOR<modo>] AS[#]n1 [LEN=n2] |
| ATARI | OPEN#n1,aux1, aux2,nom | — | — | —[2] | —[2] | — | — | — |
| CBM 64[3] | — | — | — | — | — | — | — | — |
| DRAGON | — | — | — | — | — | — | — | — |
| EQUIPOS MSX | — | FIELD#n1, n3AS<var1> | LSET/RSET | PUT#n1, <num> | GET#n1, <num> | CVI/CVS/ CVD | MKI$/MKS$/ MKD$ | OPEN<nom> [FOR<modo>] AS[#]n1 [LEN=n2] |
| HP-150 | OPEN "R",#n1, <nom>,n2 | FIELD#n1, n3AS<var1>... | LSET/RSET | PUT#n1, <num> | GET#n1, <num> | CVI/CVS/ CVD | MKI$/MKS$/ MKD$ | OPEN<nom> [FOR<modo>] AS[#]n1 [LEN=n2] |
| IBM PC | OPEN "R",#n1, <nom>,n2 | FIELD#n1, n3AS<var1>... | LSET/RSET | PUT#n1, <num> | GET#n1, <num> | CVI/CVS/ CVD | MKI$/MKS$/ MKD$ | OPEN<nom> [FOR<modo>] AS[#]n1 [LEN=n2] |
| MPF | — | — | — | — | — | — | — | — |
| NCR DM-V (MS-BASIC) | OPEN "R",#n1, <nom>,n2 | FIELD#n1, n3AS<var1>... | LSET/RSET | PUT#n1, <num> | GET#n1, <num> | CVI/CVS/ CVD | MKI$/MKS$/ MKD$ | — — |
| NEW BRAIN | — | — | — | — | — | — | — | — |
| ORIC | — | — | — | — | — | — | — | — |
| SHARP MZ-700 (MZ-BASIC) | — | — | — | — | — | — | — | — |
| SINCLAIR QL | — | — | — | — | — | — | — | — |
| SPECTRAVIDEO | — | — | LSET/RSET | PUT#n1, <num> | GET#n1, <num> | CVI/CVS/ CVD | MKI$/MKS$/ MKD$ | OPEN<nom> [FOR<modo>] AS[#]n1 [LEN=n2] |
| ZX-SPECTRUM | — | — | — | — | — | — | — | — |

[1] Para manejar estos comandos es preciso utilizar un CTRL+D e introducirlos en los argumentos de sucesivas instrucciones PRINT.

[2] Los archivos aleatorios en el ATARI con unidad de disco no funcionan de la forma comentada en el texto con los comandos PUT y GET, sino que para leer un determinado registro de un archivo, es necesario posicionarse sobre él previamente y, más tarde, realizar la operación de lectura mediante INPUT#O de escritura con PRINT#.

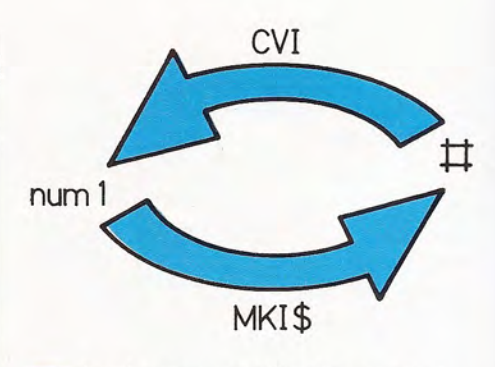
[3] Se comentará en un posterior capítulo de la obra.

```
1000 OPEN "R", #n1, "<nombre>", n2
1010 FIELD #n1, 20 AS NOM$, 15 AS AP1$,
       15 AS AP2$, 30 AS
DIR$, 10 AS CIU$, 9 A$ TEL$
1020 INPUT "QUE REGISTRO ";REG%: IF
       REG%= 0 THEN CLOSE#n1: END
1030 INPUT "NOMBRE: ";N$
1040 INPUT "APELLIDO 1: ";A1$
1050 INPUT "APELLIDO 2: ";A2$
1060 INPUT "DIRECCION :";D$
1070 INPUT "CIUDAD :";C$
1080 INPUT "TELEFONO :";T#
1100 LSET NOM$=N$
1110 LSET AP1$= A1$
1120 LSET AP2$= A2$
1130 LSET DIR$= D$
1140 LSET CIU$=C$
1150 RSET TEL$=MKD$ (T#)
1200 PUT #1,REG%
1210 GOTO 1020
```

*PROGRAMA 1: listado del programa ejemplo para la creación e introducción de datos en un archivo.*

```
1000 OPEN "R", #1, "DATOS", 99
1010 FIELD #1, 20 AS N$, 15 AS A1$,
       15 AS A2$, 30 AS D$, 10 AS C$, 9 AS T$
1020 INPUT "QUE REGISTRO ";REG%:
       IF REG%= 0 THEN CLOSE#1: END
1030 GET#1, REG%
1130 PRINT "NOMBRE: ";N$
1140 PRINT "APELLIDO 1: ";A1$
1150 PRINT "APELLIDO 2: ";A2$
1160 PRINT "DIRECCION :":D$
1170 PRINT "CIUDAD :";C$
1180 PRINT "TELEFONO :"; CVD(T$)
1210 GOTO 1020
```

*PROGRAMA 2: programa para la lectura de los datos introducidos en el archivo creado por medio del programa 1.*




*Las funciones MKI$ y CVI permiten convertir un número en una cadena de caracteres y viceversa, facilitando su almacenamiento en un espacio mínimo.*

ción del usuario una vez que el registro es extraído del archivo. Pero no hay que olvidar que los datos numéricos fueron tratados para convertirlos en alfanuméricos, por lo que ahora será preciso realizar la operación inversa. Si la función empleada para la conversión fue STR$, bastará ahora con aplicar la función VAL. Pero en el caso de que la función utilizada fuese una de las siguientes: MKI$, MKS$ o MKD$, sería preciso recurrir a las funciones CVI, CVS o CVD, respectivamente. Su cometido es el que sigue:

- CVI Convierte una cadena en un número de precisión entera.
- CVS Convierte una cadena en un número de simple precisión.
- CVD Convierte una cadena en un número de doble precisión.

Téngase en cuenta que la cadena ha de ser convertida al mismo tipo de número que constituía su origen. Es decir, si se convierte un número de precisión entera en cadena mediante el empleo de la función MKI$, no es posible reconvertir la cadena en número mediante la función CVS ni mediante la función CVD. Tan solo la función CVI nos permitirá obtener el número del que se partió.

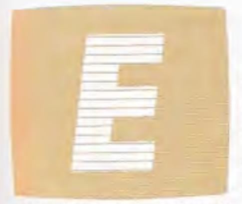
4) Cerrar el archivo para dejar libre el canal correspondiente:

```
CLOSE# n1
```

El programa ejemplo que figura junto al texto (programa 2), puede utilizarse para leer los datos que fueron introducidos en el archivo mediante el programa 1.

# Archivos aleatorios (II)

## Ejercicio práctico con un archivo de acceso directo



En este segundo capítulo dedicado a los archivos aleatorios o de acceso directo, el objetivo es profundizar en el estudio de su estructura y forma de empleo. Para el eficaz desarrollo de esta tarea se confeccionará un programa que reflejará los aspectos prácticos.

El ejemplo a desarrollar se concreta en la versión programada del archivo de una biblioteca, en el cual se introducirá información relativa a los libros existentes en la misma. Esta información constituye en su conjunto lo que denominamos archivo.

La información a almacenar puede clasificarse en diversos bloques. Cada uno de dichos bloques constituye una unidad de información denominada registro. Como ya se indicó en el primer capítulo dedicado a los archivos de acceso directo, la estructura de los registros ha de ser fija. Cada registro, a su vez, consta de una serie de campos que pueden considerarse como unidades elementales de información.




*Un ejercicio práctico como el realizado en el presente capítulo puede concluir con una útil herramienta para la gestión personal. Tal es el caso del programa confeccionado, el cual se puede utilizar para controlar el archivo bibliográfico.*

### Definición de características

La primera operación a realizar, antes de empezar el trabajo con un archivo de acceso directo, consiste en definir la estructura de los registros. Esta estructura queda determinada por tres parámetros fundamentales: número de campos que lo componen, longitud de cada uno de los campos y naturaleza de los campos. En nuestro caso, se desea crear una estructura cuyos registros contengan los siguientes campos:

- — Título.
- — Autor.
- — Editorial.
- — Número de páginas.
- — Año.
- — Código.
- — Signatura.

A continuación es imprescindible estudiar cada uno de los campos para decir cuál será la estructura y naturaleza de cada uno de ellos.

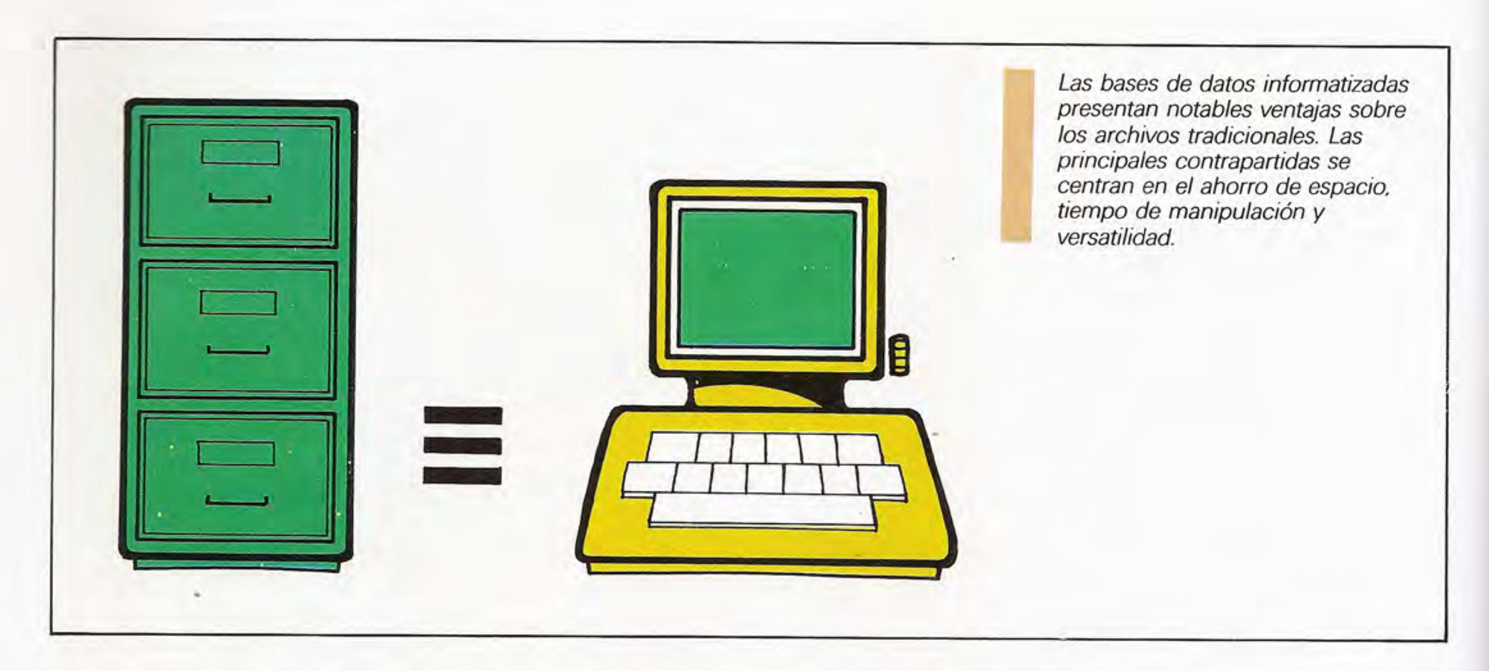
En primer lugar cabe suponer que el campo «Título» puede poseer una longitud muy variable; el número de caracteres necesarios para almacenar el título de un libro puede oscilar entre unos pocos de caracteres y 100 ó más. En todo caso, no es corriente que supere los 30 caracteres. Hay que tener en cuenta que si se otorga una longitud muy grande a este campo, surgirá el problema de que cada registro ocupará un excesivo espacio en el disco. Por el contrario, si se elige una longitud pequeña serán pocos los títulos que podrán introducirse sin proceder a su simplificación.

En definitiva, se ha de optar por una solución de compromiso intermedia entre ambas opciones. Por ejemplo, en nuestro caso podría ser razonable asignar a este campo una longitud de 40 caracteres.

Donde no cabe duda alguna es en lo que se refiere a la naturaleza del archivo: se trata de un campo alfanumérico ya que, por lo general, se compondrá de caracteres alfabéticos y, ocasionalmente, de algún número.

El siguiente campo es el de Autor. En él se desean almacenar los nombres y apellidos de los autores de los libros. Aquí surgen varias posibilidades a contemplar; la primera es que algunas veces un libro tiene más de un autor, en cuyo caso resultará difícil incluir los dos nombres en el espacio reservado para un solo nombre. En nuestro ejemplo se reservarán 30 caracteres para almacenar esta información; longitud que parece suficiente. Por supuesto, su naturaleza va a ser también alfanumérica.

El tercer campo es el destinado a la Editorial. Las características y salvedades a contemplar en este campo son muy semejantes a las ya señaladas para los campos anteriores. Ahora puede ser suficiente con una longitud de 20 caracteres para almacenar el nombre de la editorial. De nuevo, su naturaleza será alfanumérica. El número de páginas del libro es el primer campo numérico con el que tropezamos, de ahí que los problemas que plantea sean algo diferentes. Hay que decidir el tipo de variable a utilizar.



*Las bases de datos informatizadas presentan notables ventajas sobre los archivos tradicionales. Las principales contrapartidas se centran en el ahorro de espacio, tiempo de manipulación y versatilidad.*

Puede que resulte un poco extraño el hecho de tener que seleccionar el tipo de variable numérica a emplear, ya que estos campos se almacenan en forma de cadenas de caracteres. Sin embargo, la explicación es sencilla. En el primer capítulo dedicado a este tema se precisó que los números se almacenan a modo de cadenas de caracteres. Pasando por alto el empleo de las funciones STR$ y VAL que resulta poco eficiente, se dispone de las funciones del tipo MKI$, MKS$ y MKD$ y de las CVI, CVS y CVD. Las primeras permiten convertir un número en una cadena, recurriendo para ello a un formato especial que permite ahorrar memoria; el segundo bloque de funciones permite recuperar los números originales. Las funciones relacionadas se diferencian entre sí en el tipo de variables numéricas con las que trabajan. Evidentemente, es necesario conocer el tipo de variables a emplear y, por supuesto, hay que ser consecuentes con la elección y realizar las transformaciones de números a cadenas y viceversa con las funciones adecuadas.

Para almacenar el número de páginas es conveniente recurrir a variables de tipo entero; en primer lugar ello permite ahorrar memoria y, desde luego, un libro no suele tener medias hojas y tampoco suele ser tan largo como para superar las 65.000 páginas. Así pues, dicho campo se almacenará con la precisión de un número entero. El número de caracteres que emplea la función MKI$ para almacenar una variable de tipo entero es de dos bytes, por lo que la longitud del campo es de dos caracteres.

Tampoco cabe la menor duda en la elección del tipo de campo para almacenar el Año, ya que se trata de un dato numérico. Este puede almacenarse como tal en una variable entera, o bien a modo de cadena de caracteres. En el primer paso serán necesarios campos con una longitud de dos caracteres, y en el segundo de cuatro caracteres. Para el ejemplo que nos ocupa seleccionaremos el tipo numérico y, dentro de éste, la precisión de entero.

En cuanto al Código y a la Signatura, como pueden estar constituidos por caracteres alfabéticos y numéricos, será preciso almacenarlos directamente en una cadena; al efecto se reservarán 10 caracteres para cada uno de ambos campos. En resumen, los diferentes campos de cada registro, así como su naturaleza y longitudes respectivas quedan reflejados en la tabla adjunta.

Una vez definida la estructura de los campos, es necesario elegir el número de canal a emplear. Ciertamente, este no es un factor trascendente sino que tan sólo debe contemplar que su número no coincida con el de un canal ya abierto.

A la hora de abrir el canal es preciso conocer el nombre del archivo que se desea abrir y la longitud de cada uno de los campos. En nuestro caso, la orden para efectuar esta operación es la siguiente:

```
OPEN "R",#1,"DATOS",114
```

Este formato puede parecer en principio un tanto rígido; aunque no lo es tanto. Así, por ejemplo, no es necesario conocer previamente el nombre del archivo que se desea abrir; este parámetro, que en nuestro caso aparece como una constante de cadena, puede ser sustituido por una variable del mismo tipo. Lo mismo cabe afirmar de los restantes parámetros de esta instrucción.

El segundo paso a realizar es la definición de la estructura de los registros, para lo cual se utiliza la instrucción FIELD. Por ejemplo:

```
FIELD#1,40 AS títulos, 30 AS autor$, 20 AS editor$, 2 AS numpag$, 2 AS ano$, 10 AS codi$, 10 AS signat$
```

Esta especificación es necesaria, ya que para trabajar con uno de estos archivos el ordenador necesita crear una zona de memoria asociada a cada uno de los canales. Esta zona de memoria temporal (buffer) actúa como intermediario para la transferencia de información entre la memoria del ordenador y el archivo externo. En el buffer están reservadas zonas para cada uno de los campos que se van a emplear. Cada una

de estas zonas puede manejarse de forma independiente, lo cual significa que pueden introducirse los datos que se deseen y en el orden que parezca más conveniente.

## Creando un programa para manejar archivos de acceso directo

Una vez estudiado el método para la apertura del archivo y decidida su estructura, se está en condiciones de crear un programa para su tratamiento.

Antes de empezar la codificación del programa, es necesario decidir qué operaciones deseamos que éste sea capaz de realizar.

En nuestro caso, crearemos un programa que realice las operaciones básicas en un archivo de acceso directo; estas operaciones son las de creación, actualización y consulta.

La estructura de nuestro programa consistirá en una zona central, encargada de la apertura del archivo y de la selección de la operación a realizar, y de un surtido de subrutinas especializadas.

Cada operación a realizar será ejecutada por una subrutina distinta creada al efecto; tras ejecutarse, la secuencia del programa volverá a la zona de selección de las siguientes operaciones a realizar.

A continuación se confeccionará el cuerpo principal del programa que incluye el menú para la selección de la opción deseada.

Las dos primeras líneas del programa (1000 y 1010) abren el archivo y crean el buffer de trabajo:

```
1000 OPEN"R",#1,"DATOS",114
1010 FIELD#1, 40 AS tiulo$, 30 AS autor$, 20 AS editor$,
     2 AS numpag$, 2 AS ano$, 10 AS codi$, 10 AS signat$
```

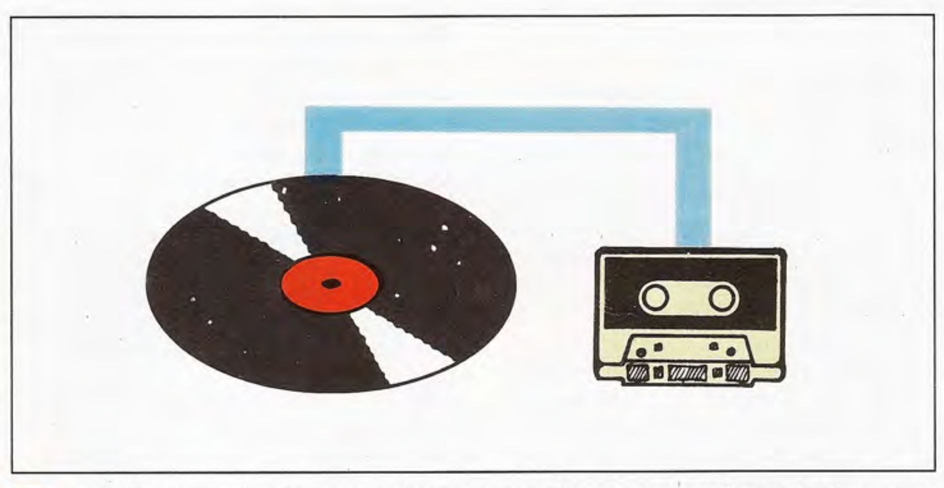
Acto seguido llega el momento de borrar la pantalla e inicializar las variables NUM% y NMAX% cuya misión es actuar como contadores. En el caso de que el ordenador sea del tipo IBM-PC o MSX, también será preciso eliminar de la pantalla los recordatorios de las funciones asignadas a las teclas de función. Todo ello corre a cargo de las siguientes líneas de programa:



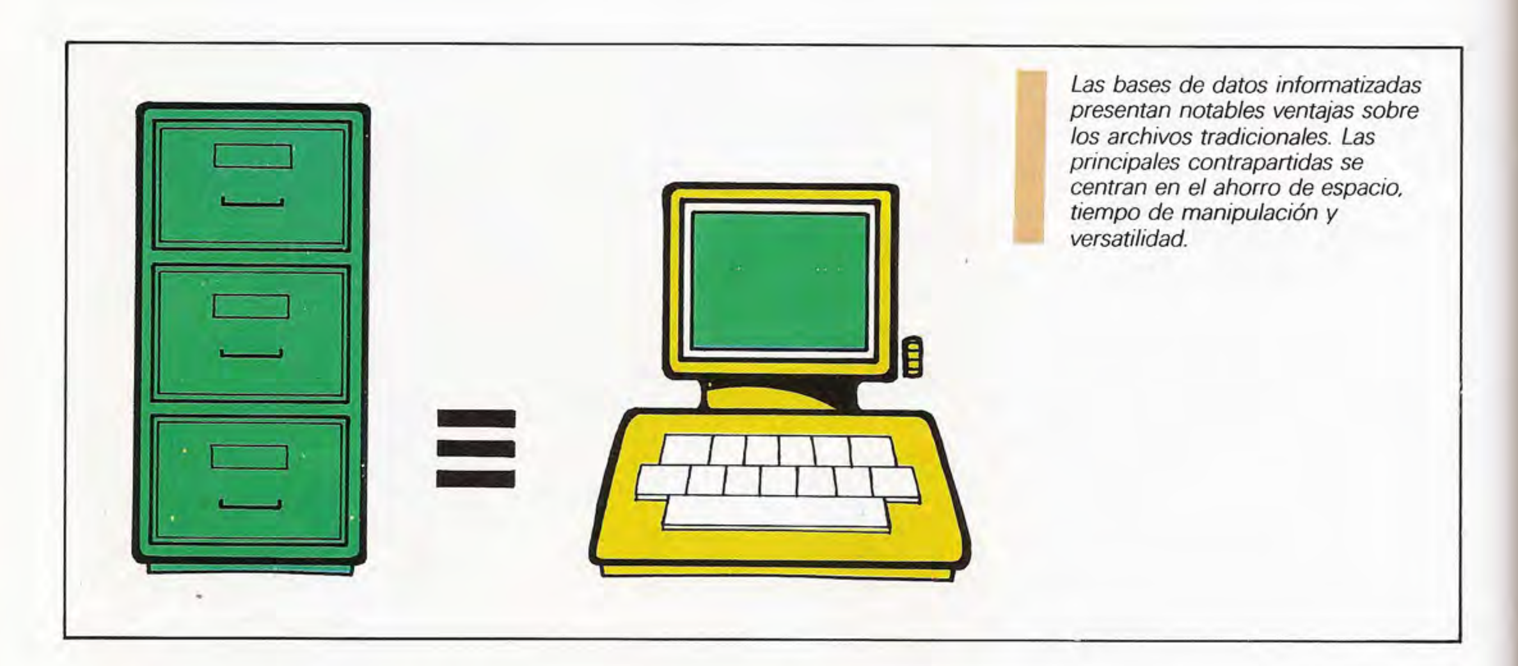
*La estructura de un archivo de acceso directo guarda un cierto paralelismo con la organización de los archivos tradicionales. La información se agrupa en bloques denominados registros y éstos, a su vez, engloban a unconjunto de campos.*

```
1020 NUM%=0 : NMAX=0
1900 CLS : KEY OFF
```

La zona comprendida entre las líneas 2000 a la 2040 presentan por pantalla el menú de selección. La visualización de las opciones se realiza mediante una serie de instrucciones PRINT, antepo-



*Las diferencias de acceso que presenta un archivo de acceso directo frente a un archivo secuencial son perceptibles en las diferencias que se manifiestan entre un disco de audio y una casete.*



*Las bases de datos informatizadas presentan notables ventajas sobre los archivos tradicionales. Las principales contrapartidas se centran en el ahorro de espacio, tiempo de manipulación y versatilidad.*

Puede que resulte un poco extraño el hecho de tener que seleccionar el tipo de variable numérica a emplear, ya que estos campos se almacenan en forma de cadenas de caracteres. Sin embargo, la explicación es sencilla. En el primer capítulo dedicado a este tema se precisó que los números se almacenan a modo de cadenas de caracteres. Pasando por alto el empleo de las funciones STR$ y VAL que resulta poco eficiente, se dispone de las funciones del tipo MKI$, MKS$ y MKD$ y de las CVI, CVS y CVD. Las primeras permiten convertir un número en una cadena, recurriendo para ello a un formato especial que permite ahorrar memoria; el segundo bloque de funciones permite recuperar los números originales. Las funciones relacionadas se diferencian entre sí en el tipo de variables numéricas con las que trabajan. Evidentemente, es necesario conocer el tipo de variables a emplear y, por supuesto, hay que ser consecuentes con la elección y realizar las transformaciones de números a cadenas y viceversa con las funciones adecuadas.

Para almacenar el número de páginas es conveniente recurrir a variables de tipo entero; en primer lugar ello permite ahorrar memoria y, desde luego, un libro no suele tener medias hojas y tampoco suele ser tan largo como para superar las 65.000 páginas. Así pues, dicho campo se almacenará con la precisión de un número entero. El número de caracteres que emplea la función MKI$ para almacenar una variable de tipo entero es de dos bytes, por lo que la longitud del campo es de dos caracteres.

Tampoco cabe la menor duda en la elección del tipo de campo para almacenar el Año, ya que se trata de un dato numérico. Este puede almacenarse como tal en una variable entera, o bien a modo de cadena de caracteres. En el primer paso serán necesarios campos con una longitud de dos caracteres, y en el segundo de cuatro caracteres. Para el ejemplo que nos ocupa seleccionaremos el tipo numérico y, dentro de éste, la precisión de entero.

En cuanto al Código y a la Signatura, como pueden estar constituidos por caracteres alfabéticos y numéricos, será preciso almacenarlos directamente en una cadena; al efecto se reservarán 10 caracteres para cada uno de ambos campos. En resumen, los diferentes campos de cada registro, así como su naturaleza y longitudes respectivas quedan reflejados en la tabla adjunta.

Una vez definida la estructura de los campos, es necesario elegir el número de canal a emplear. Ciertamente, este no es un factor trascendente sino que tan sólo debe contemplar que su número no coincida con el de un canal ya abierto.

A la hora de abrir el canal es preciso conocer el nombre del archivo que se desea abrir y la longitud de cada uno de los campos. En nuestro caso, la orden para efectuar esta operación es la siguiente:

```
OPEN "R",#1,"DATOS",114
```

Este formato puede parecer en principio un tanto rígido; aunque no lo es tanto. Así, por ejemplo, no es necesario conocer previamente el nombre del archivo que se desea abrir; este parámetro, que en nuestro caso aparece como una constante de cadena, puede ser sustituido por una variable del mismo tipo. Lo mismo cabe afirmar de los restantes parámetros de esta instrucción.

El segundo paso a realizar es la definición de la estructura de los registros, para lo cual se utiliza la instrucción FIELD. Por ejemplo:

```
FIELD#1,40 AS títulos, 30 AS autor$, 20 AS editor$, 2 AS numpag$, 2 AS ano$, 10 AS codi$, 10 AS signat$
```

Esta especificación es necesaria, ya que para trabajar con uno de estos archivos el ordenador necesita crear una zona de memoria asociada a cada uno de los canales. Esta zona de memoria temporal (buffer) actúa como intermediario para la transferencia de información entre la memoria del ordenador y el archivo externo. En el buffer están reservadas zonas para cada uno de los campos que se van a emplear. Cada una

de estas zonas puede manejarse de forma independiente, lo cual significa que pueden introducirse los datos que se deseen y en el orden que parezca más conveniente.

## Creando un programa para manejar archivos de acceso directo

Una vez estudiado el método para la apertura del archivo y decidida su estructura, se está en condiciones de crear un programa para su tratamiento.

Antes de empezar la codificación del programa, es necesario decidir qué operaciones deseamos que éste sea capaz de realizar.

En nuestro caso, crearemos un programa que realice las operaciones básicas en un archivo de acceso directo; estas operaciones son las de creación, actualización y consulta.

La estructura de nuestro programa consistirá en una zona central, encargada de la apertura del archivo y de la selección de la operación a realizar, y de un surtido de subrutinas especializadas.

Cada operación a realizar será ejecutada por una subrutina distinta creada al efecto; tras ejecutarse, la secuencia del programa volverá a la zona de selección de las siguientes operaciones a realizar.

A continuación se confeccionará el cuerpo principal del programa que incluye el menú para la selección de la opción deseada.

Las dos primeras líneas del programa (1000 y 1010) abren el archivo y crean el buffer de trabajo:

```
1000 OPEN"R",#1,"DATOS",114
1010 FIELD#1, 40 AS tiulo$, 30 AS autor$, 20 AS editor$,
     2 AS numpag$, 2 AS ano$, 10 AS codi$, 10 AS signat$
```

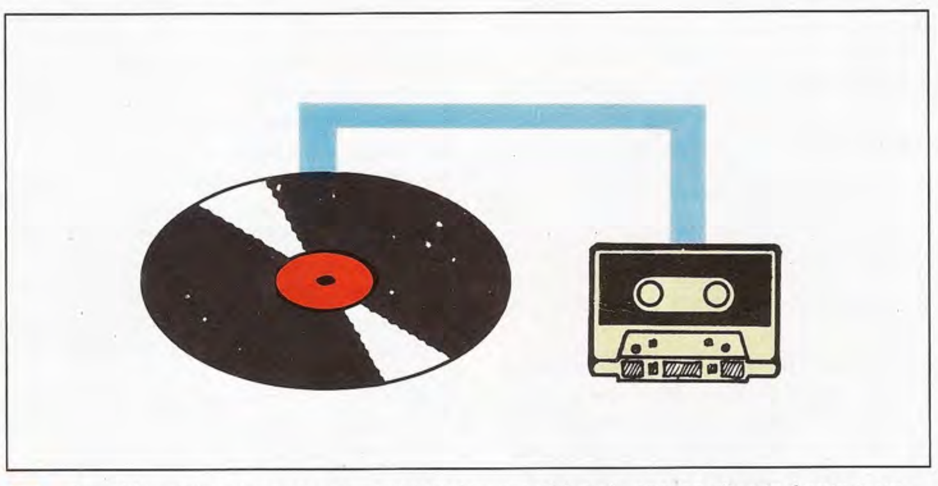
Acto seguido llega el momento de borrar la pantalla e inicializar las variables NUM% y NMAX% cuya misión es actuar como contadores. En el caso de que el ordenador sea del tipo IBM-PC o MSX, también será preciso eliminar de la pantalla los recordatorios de las funciones asignadas a las teclas de función. Todo ello corre a cargo de las siguientes líneas de programa:



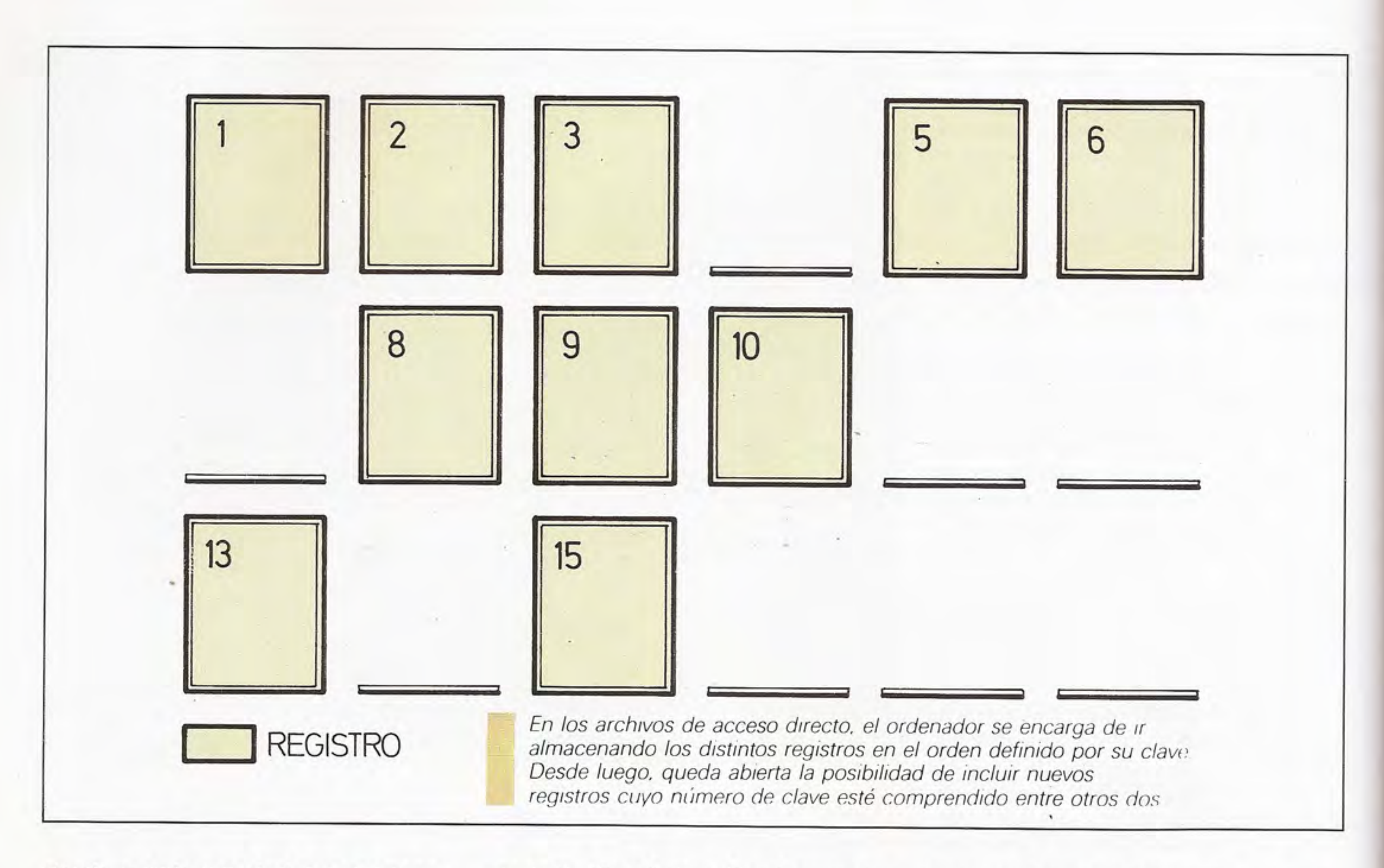
*La estructura de un archivo de acceso directo guarda un cierto paralelismo con la organización de los archivos tradicionales. La información se agrupa en bloques denominados registros y éstos, a su vez, engloban a unconjunto de campos.*

```
1020 NUM%=0 : NMAX=0
1900 CLS : KEY OFF
```

La zona comprendida entre las líneas 2000 a la 2040 presentan por pantalla el menú de selección. La visualización de las opciones se realiza mediante una serie de instrucciones PRINT, antepo-



*Las diferencias de acceso que presenta un archivo de acceso directo frente a un archivo secuencial son perceptibles en las diferencias que se manifiestan entre un disco de audio y una casete.*



*En los archivos de acceso directo, el ordenador se encarga de ir almacenando los distintos registros en el orden definido por su clave. Desde luego, queda abierta la posibilidad de incluir nuevos registros cuyo número de clave esté comprendido entre otros dos*

niendo a cada una de las operaciones que se pueden realizar un número que permitirá seleccionarlas:

```
1950 PRINT
2000 PRINT TAB(28);"MENU DE SELECCION" : PRINT :
     PRINT
2010 PRINT TAB(25);"1— CREACION DEL ARCHIVO" :
     PRINT
2020 PRINT TAB(25);"2— LECTURA DE UN REGISTRO" :
     PRINT
2030 PRINT TAB(25);"3— MODIFICACION DE UN REGIS-
     TRO" : PRINT
2040 PRINT TAB(25);"4— LISTADO DEL ARCHIVO" : PRINT
```

Con este menú a la vista, el usuario debe seleccionar la operación que desea realizar. Para que tal selección sea efectiva sin más que pulsar una tecla, es necesario recurrir a la función INKEY$.

Nuestro programa recoge el carácter pulsado en la variable B$ y, acto seguido, se comprueba si dicha variable tiene algún contenido. De no poseer contenido alguno puede extraerse la conclusión de que no se ha pulsado ninguna tecla, por lo que es preciso muestrear de nuevo el teclado hasta que se pulse una tecla. En tal situación el programa queda encerrado dentro de un bucle del que no saldrá hasta que el usuario seleccione una opción.

```
2050 B$=INKEY$ : IF B$="" THEN GOTO 2050
```

Las líneas que van de la 2060 a la 2080 gestionan la selección realizada. Al efecto, cada vez que se pulsa una tecla, el programa analiza si ésta corresponde a una de las posibles opciones. En caso negativo, se vuelve a examinar de nuevo el teclado.

Si la tecla pulsada corresponde a una de la posibles opciones, la instrucción ON GOSUB de la línea 2080 ordenará el salto a la rutina adecuada.

La línea 2090 provoca el retorno al menú una vez que concluye la ejecución de la subrutina a la que se ha bifurcado.

```
2060 caracter=ASC(B$)—48
2070 IF caracter < 1 OR caracter > 4 THEN GOTO 2050
2080 ON caracter GOSUB 10000, 20000, 30000, 40000
2090 GOTO 1900
```

Aquí concluye el cuerpo principal del programa. A partir de este punto se encuentran las subrutinas encargadas de procesar las distintas opciones.

Si se elige la opción 1 (creación del archivo), se ejecutará la siguiente rutina que solicita la introducción de los datos para cada registro:

```
10000 CLS : INPUT "TITULO DEL LIBRO"; T$
10020 IF T$="FIN" THEN RETURN
10050 INPUT "AUTOR"; A$
10100 INPUT "EDITOR"; E$
10150 INPUT "NUMERO DE PAGINAS"; N$
10200 INPUT "ANO"; Y$
10250 INPUT "CODIGO"; C$
10300 INPUT "SIGNATURA"; S$
```

A continuación esto datos deben ser transferidos al buffer, realizando, al mismo tiempo, la operación de ajustar cada dato a las características de su respectivo campo; ello supone adecuar el dato al tipo de variable que se utilice:

```
10310 LSET titulo$=T$
10320 LSET autor$=A$
10330 LSET editor$=E$
10340 RSET numpag$=MKI$(VAL(N$))
10350 RSET ano$=MKI$(VAL(Y$))
```

```
10360 LSET codi$=C$
10370 LSET signa$=S$
```

Es importante ir actualizando el contador que sirve para conocer qué posición relativa ocupa el registro dentro del archivo que se está creando. Dicha posición viene dada por el valor de la variable NUM%:

```
10380 NUM%=NUM%+1
```

También es preciso saber cuántos registros han sido creados para, por ejemplo, cuando sea necesario listarlos, realizar la operación sin que se produzca un error:

```
10390 NMAX%=NMAX%+1
```

Para transferir el contenido del registro a la posición deseada del archivo se emplea una instrucción PUT. En ella se especifica el número de canal asociado al archivo en cuestión y la posición del registro dentro del archivo:

```
10400 PUT#1, NUM%
```

Tras ello hay que pasar a la fase de introducción de otro registro, de lo cual se ocupa la instrucción 10410.

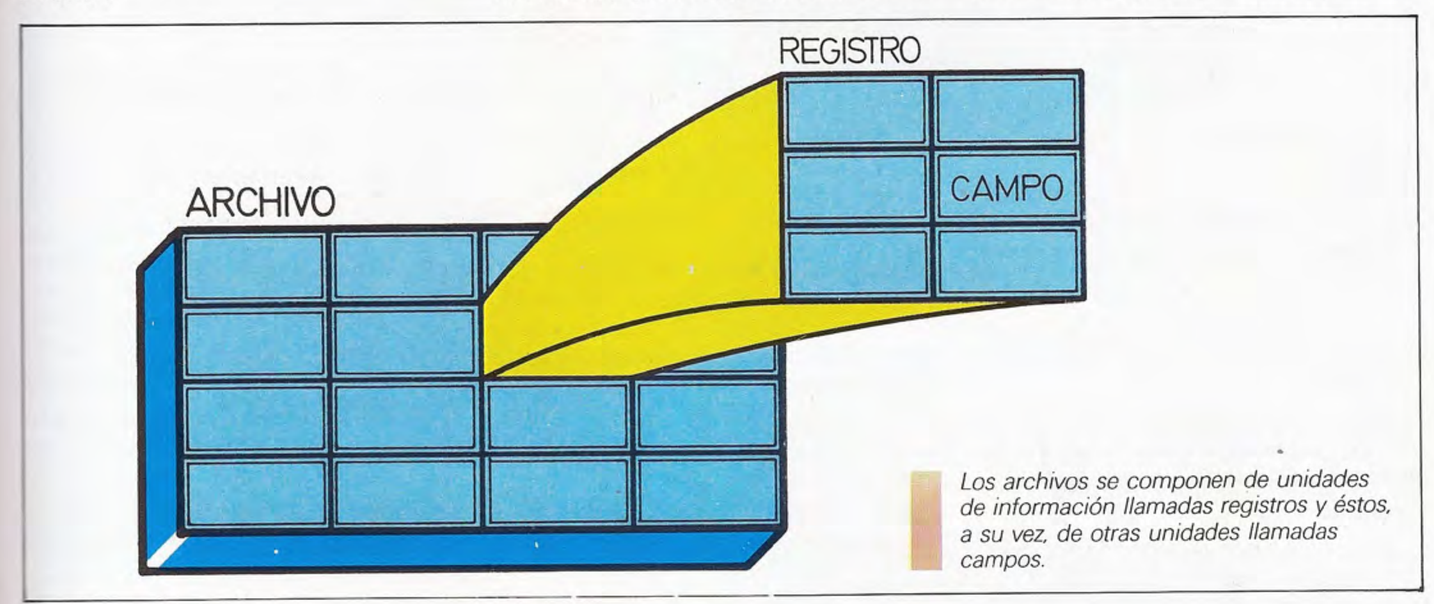
A la vista de las instrucciones descritas puede parecer que esta rutina no tiene salida, y que constituye un bucle cerrado por completo. Sin embargo se ha previsto una salida.

Esta se encuentra en la línea 10020; en ella se examina el campo de título sobre la variable T$, y en caso de que se haya introducido la palabra Fin, la secuencia de ejecución volverá al menú principal del programa.

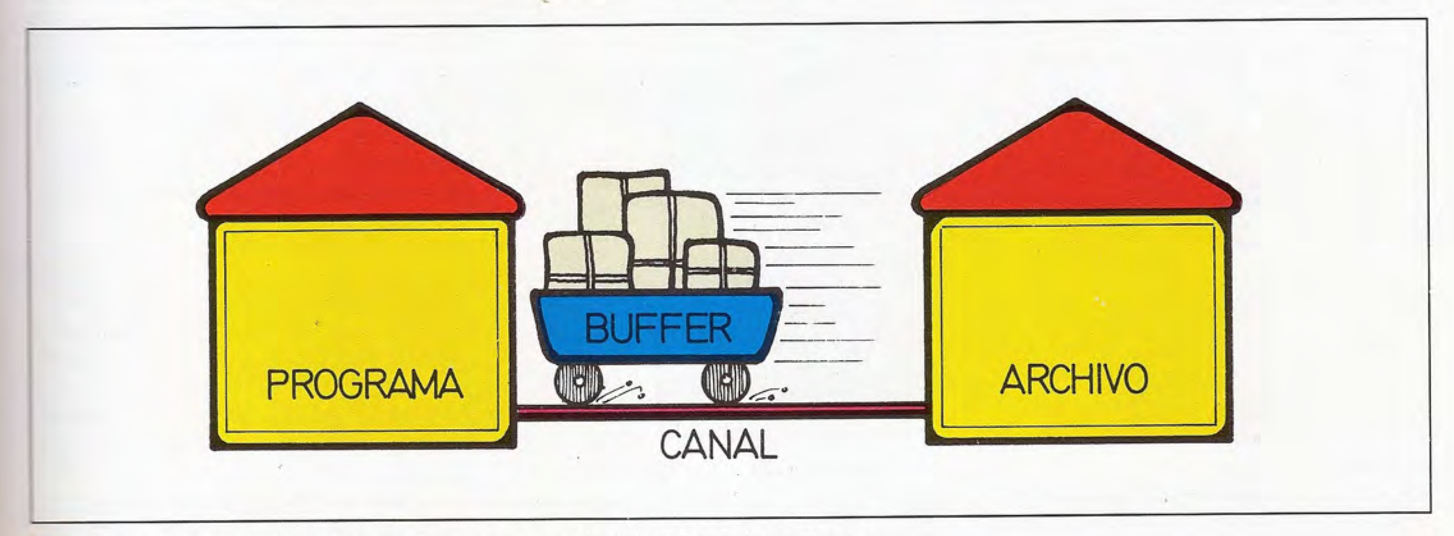
```
10410 GOTO 10000
10420 RETURN
```

En la línea 10420 concluye con el comando RETURN la subrutina de creación del archivo.

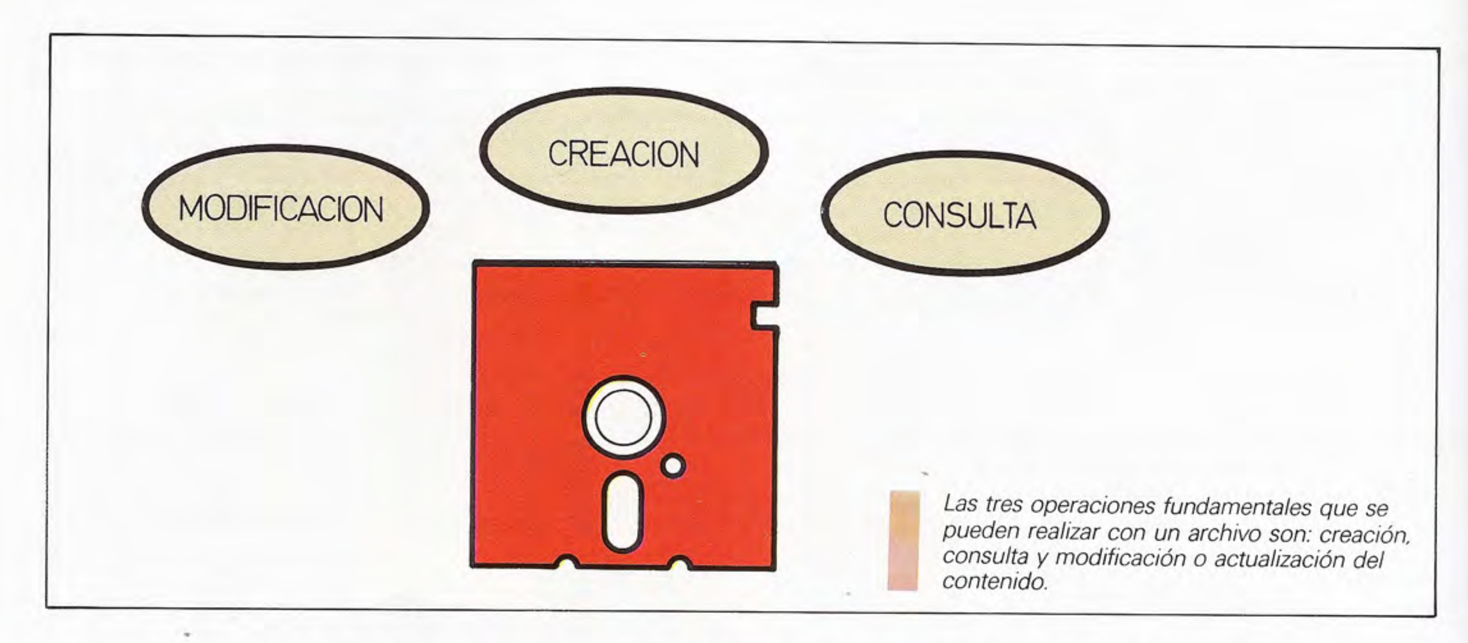
La próxima subrutina a crear es la de consulta de registros del archivo, la cual




*Los archivos se componen de unidades de información llamadas registros y éstos, a su vez, de otras unidades llamadas campos.*




*El canal actúa a modo de ente intermedio a través del cual los datos pueden pasar del programa o proceso en curso hacia el archivo.*




Las tres operaciones fundamentales que se pueden realizar con un archivo son: creación, consulta y modificación o actualización del contenido.

está asociada a la opción Z del menú principal.

La primera opción que debe realizar esta subrutina es la de consultar al usuario cuál es el registro que se desea leer:

```
20000 CLS : INPUT "QUE REGISTRO DESEAS LEER";
      NUM%
```

A continuación, y una vez que se conoce el registro que se desea consultar, se ejecutará una instrucción GET, la cual accede al disco y lee el registro cuyo número se ha solicitado:

```
20010 GET#1, NUM%
```

Y por fin, una vez que el registro a pasado al buffer, la siguiente operación es mostrar por pantalla el contenido del mismo. Esta tarea se puede realizar sencillamente mediante una secuencia de instrucciones PRINT:

```
20020 PRINT "1—TITULO"; titulo$
20030 PRINT "2—AUTOR"; autor$
20040 PRINT "3—EDITOR"; editor$
20050 PRINT "4—NUMERO DE PAGINAS"; CVI (numpag$)
20060 PRINT "5—ANO"; CVI(ano$)
20070 PRINT "6—CODIGO"; codi$
20080 PRINT "7—SIGNATURA"; signa$
20090 LET B$=INKEY$ : IF B$="" THEN GOTO 20090
```

Tras presentar en la pantalla el contenido del registro, hay que regresar al programa principal por medio del correspondiente RETURN:

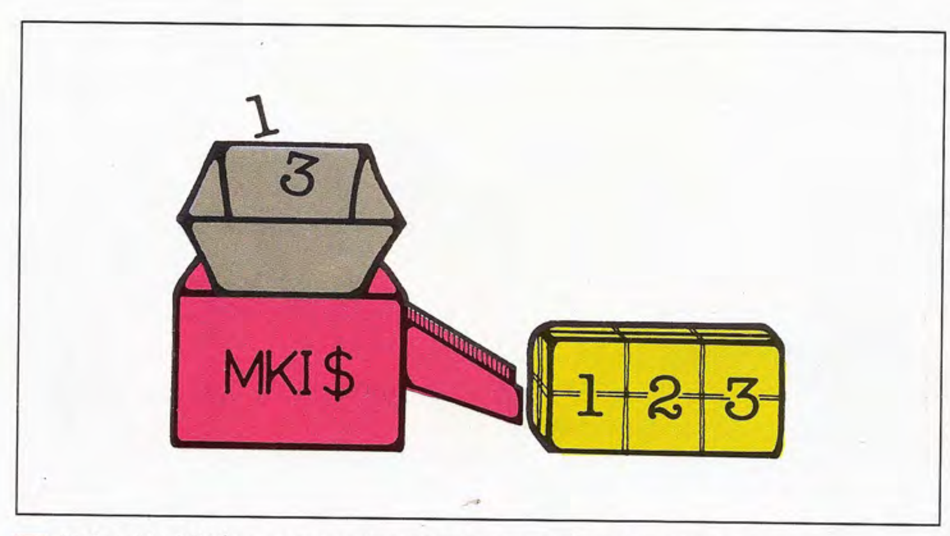
```
20100 RETURN
```

La subrutina asociada a la opción 3 (modificación de un registro), muestra en pantalla el contenido del registro que se solicite a través de una llamada a la subrutina de lectura de un registro.

El salto a la subrutina de lectura se produce al ejecutar la instrucción GOSUB 20010 de la línea 30010.

Acto seguido se ofrece al usuario la posibilidad de renunciar a la modificación de campos, sin más que responder con una acción sobre la tecla 8. En todo caso, la situación habitual en este punto será la de responder a la pregunta "QUE CAMPO DESEAS MODIFICAR" con el número del campo oportuno. Este número, almacenado en la variable C, será utilizado para la instrucción ON GOSUB de la línea 30030 para bifurcar la secuencia de ejecución hacia la zona adecuada.

La zona de programa que sigue coincide con la asociada al tratamiento de la opción descrita:




La función MKI$ se encarga de empaquetar un número en una cadena, permitiendo con ello ahorrar memoria frente al uso de la función STR$

```
30000 CLS : INPUT "QUE REGISTRO DESEAS MODIFI
      CAR"; NUM%
30010 GOSUB 20010
30015 PRINT "SI NO DESEAS MODIFICAR NINGUNO PUL
      SA 8"
30020 INPUT "QUE CAMPO DESEAS MODIFICAR"; C
30030 ON C GOSUB 30100, 30200, 30300, 30400, 30500,
      30600, 30700
30040 RETURN
30100 INPUT "TITULO"; T$
30110 LSET titulo$=T$
30120 PUT#1, NUM% : RETURN
30200 INPUT "AUTOR"; A$
30210 LSET autor$=A$
30220 PUTK1, NUM% : RETURN
30300 INPUT "EDITOR"; E$
30310 LSET editor$=E$
30320 RETURN
30400 INPUT "NUMERO DE PAGINAS"; N$
30410 RSET numpag$=N$
30420 RETURN
30500 INPUT "ANO"; Y$
30510 RSET ano$=Y$
30520 RETURN
30600 INPUT "CODIGO"; C$
30610 LSET codi$=C$
30620 RETURN
30700 INPT "SIGNATURA"; S$
30710 LSET signa$=S$
30720 RETURN
```

La cuarta y última opción del menú da paso al listado de un archivo. Su puesta en práctica se fundamenta en sucesivas llamadas a la subrutina de lectura de un registro. Ello tiene lugar dentro del siguiente bucle:

```
40000 CLS : FOR NUM%=1 TO NMAX%
40010 GOSUB 20010
40020 NEXT I
40030 RETURN
```

El listado adjunto muestra el progra-

| ESTRUCTURA DE CADA REGISTRO | | |
|---|---|---|
| **Nombre del campo** | **Número de caracteres** | **Tipo de campo** |
| Título | 40 | Alfanumérico |
| Autor | 30 | Alfanumérico |
| Editorial | 20 | Alfanumérico |
| Número pág. | 2 | Número entero |
| Año | 2 | Número entero |
| Código | 10 | Alfanumérico |
| Signatura | 10 | Alfanumérico |
| Número total de caracteres en cada registro: 114 | | |



*Pantalla de entrada al programa con el menú de opciones. Su presentación corre a cargo de la rutina o cuerpo principal del programa.*



*Pantalla asociada a la opción de modificación de un registro. Los contenidos que aparecen tras los diversos campos proceden de un ejemplo de ficha bibliográfica.*

```
1000 OPEN"R",#1, "DATOS", 114
1010 FIELD#1, 40 AS titulo$, 30 AS autor$, 20 AS editor$,
2 AS numpag$, 2 AS ano$, 10 AS codi$, 10 AS signat$
1020 NUM%=0 : NMAX=0
1900 CLS : KEY OFF
1950 PRINT
2000 PRINT TAB(28); "MENU DE SELECCION" : PRINT : PRINT
2010 PRINT TAB(25); "1- CREACION DEL ARCHIVO" : PRINT
2020 PRINT TAB(25); "2- LECTURA DE UN REGISTRO" : PRINT
2030 PRINT TAB(25); "3- MODIFICACION DE UN REGISTRO" : PRINT
2040 PRINT TAB(25); "4- LISTADO DEL ARCHIVO" : PRINT
2050 B$=INKEY$ : IF B$=" " THEN GOTO 2050
2060 caracter=ASC(B$)-48
2070 IF caracter < 1 OR caracter > 4 THEN GOTO 2050
2080 ON caracter GOSUB 10000, 20000, 30000, 40000
2090 GOTO 1900
10000 CLS : INPUT "TITULO DEL LIBRO"; T$
10020 IF T$="FIN" THEN RETURN
10050 INPUT "AUTOR"; A$
10100 INPUT "EDITOR"; E$
10150 INPUT "NUMERO DE PAGINAS"; N$
10200 INPUT "ANO"; Y$
10250 INPUT "CODIGO"; C$
10300 INPUT "SIGNATURA"; S$
10310 LSET titulo$=T$
10320 LSET autor$=A$
10330 LSET editor$=E$
10340 RSET numpag$=MKI$(VAL(N$))
10350 RSET ano$=MKI$(VAL(Y$))
10360 LSET codi$=C$
10370 LSET signa$=S$
10380 NUM%=NUM%+1
10390 NMAX%=NMAX%+1
10400 PUT#1, NUM%
10410 GOTO 10000
10420 RETURN
20000 CLS : INPUT "QUE REGISTRO DESEAS LEER"; NUM%
20010 GET#1, NUM%
20020 PRINT "1-TITULO"; titulo$
20030 PRINT "2-AUTOR"; autor$
20040 PRINT "3-EDITOR"; editor$
20050 PRINT "4-NUMERO DE PAGINAS"; CVI(numpag$)
20060 PRINT "5-ANO"; CVI(ano$)
20070 PRINT "6-CODIGO"; codi$
20080 PRINT "7-SIGNATURA"; signa$
20090 LET B$=INKEY$ : IF B$=" " THEN GOTO 20090
20100 RETURN
30000 CLS : INPUT "QUE REGISTRO DESEAS MODIFICAR"; NUM%
30010 GOSUB 20010
30015 PRINT "SI NO DESEAS MODIFICAR NINGUNO PULSA 8"
30020 INPUT "QUE CAMPO DESEAS MODIFICAR"; C
30030 ON C GOSUB 30100, 30200, 30300, 30400, 30500, 30600, 30700
30040 RETURN
30100 INPUT "TITULO"; T$
30110 LSET titulo$=T$
30120 PUT#1, NUM% : RETURN
30200 INPUT "AUTOR"; A$
30210 LSET autor$=A$
30220 PUT#1, NUM% : RETURN
30300 INPUT "EDITOR"; E$
30310 LSET editor$=E$
30320 RETURN
30400 INPUT "NUMERO DE PAGINAS"; N$
30410 RSET numpag$=N$
30420 RETURN
30500 INPUT "ANO"; Y$
30510 RSET ano$=Y$
30520 RETURN
30600 INPUT "CODIGO"; C$
30610 LSET codi$=C$
30620 RETURN
30700 INPUT "SIGNATURA"; S$
30710 LSET signa$=S$
30720 RETURN
40000 CLS: FOR NUM%=1 TO NMAX%
40010 GOSUB 20010
40020 NEXT I
40030 RETURN
```

*Listado del PROGRAMA.*

ma en su totalidad. Tal como cabe deducir de las descripciones realizadas, la creación y tratamiento de archivos aleatorios o de acceso directo no presenta mayores complicaciones de las derivadas del propio conocimiento de los comandos y funciones BASIC especializadas en su manipulación. Al respecto, cabe precisar que en el ejemplo propuesto se ha utilizado un repertorio específico que estará sujeto a variaciones según el dialecto BASIC que equipe al ordenador utilizado. Por ello, antes de proceder a la instrucción de este programa, habrá que revisar la equivalencia de comandos y funciones de su dialecto BASIC con respecto al repertorio empleado en nuestro ejemplo práctico.

# Agenda telefónica

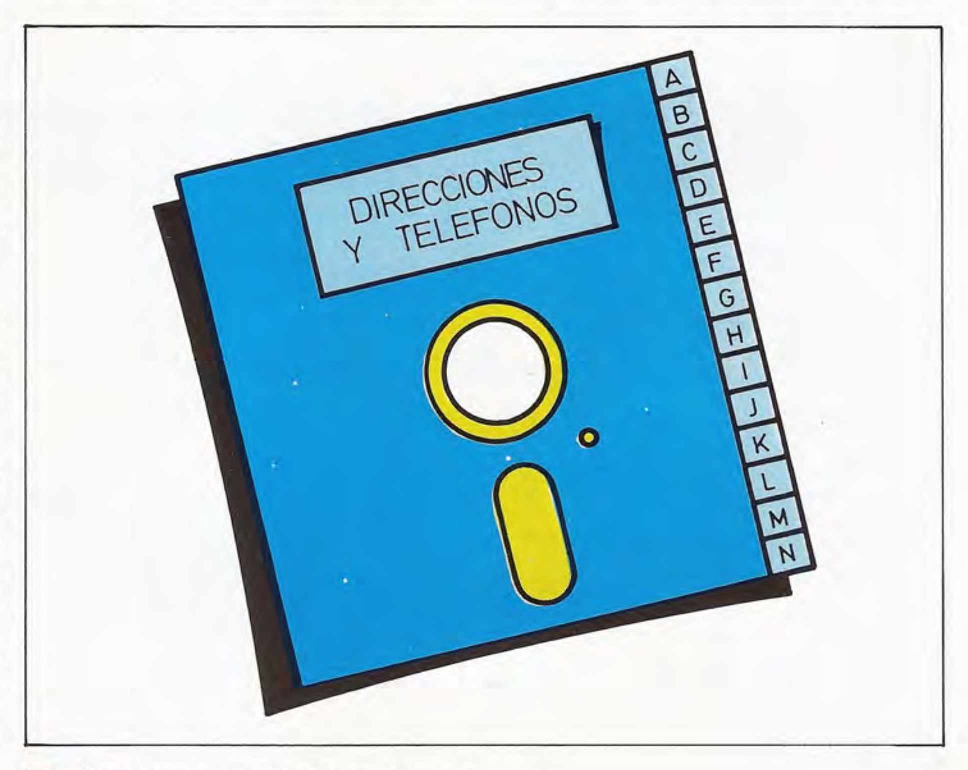
## Afianzando conceptos



En anteriores capítulos de la obra se ha hablado largo y tendido del tratamiento que pueden recibir los archivos aleatorios o de acceso directo. En esta ocasión se va a hablar de nuevo de ellos, aunque desde una perspectiva totalmente práctica; concretamente, desarrollando un programa de aplicación que resultará de plena utilidad.

Al tiempo que se confecciona el programa, se expondrán las peculiaridades que en este aspecto presenta el BASIC de los equipos ATARI; dialecto éste que se adoptará como herramienta de programación. Por lo tanto, a lo largo de los próximos párrafos se introducirán y estudiarán con detalles una serie de nuevos comandos relativos al manejo de los archivos en disco, específicos de estas máquinas.

Como ya se ha precisado en otras ocasiones, los poseedores de máquinas dotadas de otra versión del lenguaje BASIC no deben echar en saco roto este capítulo. La traducción al BASIC respectivo de cada ordenadore se podrá realizar muy fácilmente, sin más que establecer la equivalencia de unos pocos comandos, según se indica en las diferentes tablas de conversión publicadas a lo largo de la obra.




*Con el programa descrito en el texto se convierte al ordenador en una fiel y útil agenda de direcciones y teléfonos.*

### Una agenda informatizada

El programa-ejemplo a codificar coincide con una agenda de direcciones y teléfonos. La analogía de la agenda convencional con los archivos informáticos es enorme. Así, una agenda constituye en sí un archivo, entendiendo por archivo a un conjunto de informaciones relacionadas entre sí.

La agenda, como conjunto, es divisible en otras partes que son las diferentes anotaciones para cada persona o entidad, lo que coincide totalmente con el concepto informático de registro. Estas anotaciones de la agenda estarán compuestas, a su vez, por el nombre, la dirección,ciudad, el teléfono... Elementos que constituyen lo que en términos informáticos se denominan campos de un registro.

En definitiva, nuestra agenda informatizada estará almacenada en un archivo (en este caso en disco, para poder utilizar el acceso aleatorio), dividido en registros que contendrán los datos relativos a una persona depositados en los respectivos campos.

Antes de seguir adelante, hay que establecer un número de campos que se van a necesitar, así como su longitud, para poder ir definiendo en consecuencia la estructura real de la agenda informática.

Un primer campo podría ser el reservado al nombre de cada persona. En él se incluirá tanto el nombre de pila como los respectivos apellidos. Este campo va a tener una importancia fundamental en nuestro programa, ya que para evitar que éste se complique en exceso y resulte demasiado extenso, el campo del nombre va a servir de índice; o lo que es lo mismo, de clave interna para el acceso a la información guardada en la agenda. Los restantes campos pueden ser los siguientes:

— Campo de dirección (calle, número, piso, etc.).
— Campo de ciudad o localidad.
— Campo de provincia y código postal.
— Campo de número de teléfono (incluyendo prefijos, etc.).

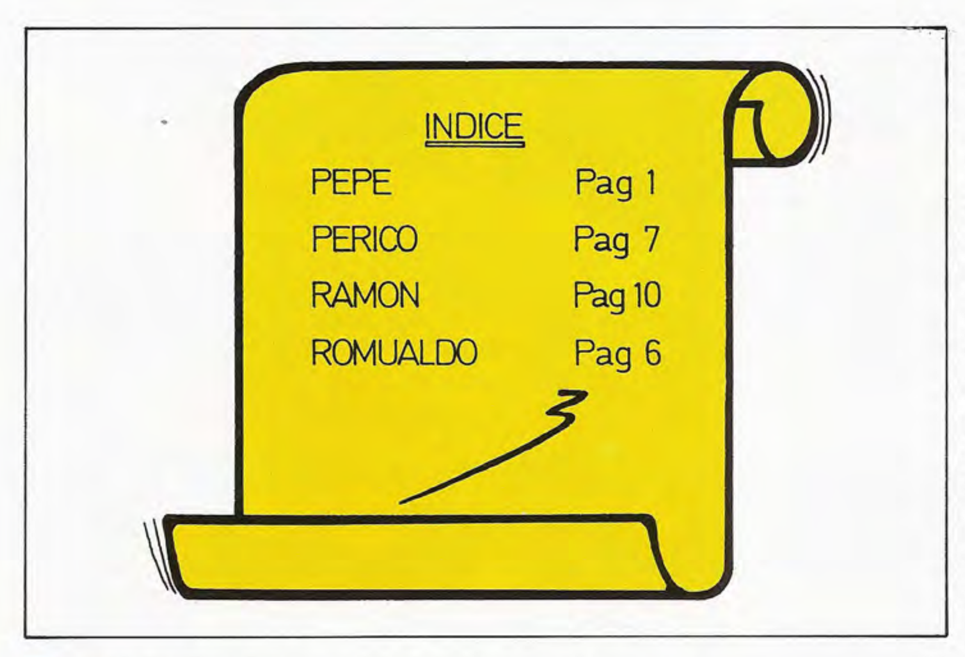
La longitud de estos campos se puede elegir libremente, ya que no está prefijado ningún valor a priori.

El siguiente paso es establecer las distintas funciones que debe gestionar el programa de agenda. Por ejemplo:

— Búsqueda de un registro específico.
— Introducción de un nuevo registro.
— Listado de todos los registros.




*Las matrices multidimensionales de caracteres se pueden simular en el ATARI «troceando» artificialmente una cadena suficientemente larga.*




*El índice creado con la agenda es el que permitirá el acceso a toda la información contenida en ella.*

— Modificación de todos los registros.
— Borrado de un registro.
— Abandonar el programa, almacenando los datos necesarios.

Estas seis funciones encierran todo el proceso de creación, actualización y almacenamiento del archivo que constituirá la agenda. A partir de ellas se puede ya definir una primera estructura del programa, que va a ser controlado por un menú de opciones coincidentes con las funciones indicadas. A partir de este menú y de la opción elegida en cada momento, el programa bifurcará hacia una serie de subrutinas que, una vez ejecutadas, devolverán el control al menú de opciones. Dicho menú constituirá el programa principal o bucle principal, que se ejecutará indefinidamente, hasta que se dé paso a la opción de salida del programa.

Se puede ya empezar a codificar el programa; comenzando por una zona de presentación que consistirá, sencillamente, en un título centrado mediante el comando POSITION. Esta es la tarea asignada a las instrucciones 10, 20 y 30.

Previamente a la codificación del bucle principal del programa, habrá que realizar una subrutina de inicialización del programa y sus variables; subrutina que será llamada por la línea 40 de la zona de presentación.

La subrutina de inicialización sólo será ejecutada una vez al principio del programa, por lo que podría haber sido escrita directamente en las líneas 30 y 50 de la presentación. No obstante, se ha preferido darle la estructura de subrutina ya que así queda más claro el programa y se ve facilitada la comprobación de su correcto funcionamiento. La subrutina de inicialización se ocupará de dimensionar y de inicializar las diferentes variables a utilizar en el programa, así como de obtener del disco la información necesaria para que sea posible acceder a la agenda con posterioridad. Por este motivo, se pospondrá la confección de dicha subrutina hasta el momento en el que se hayan definido las variable necesarias, así como la estructura de los ficheros en disco. Hasta que llegue ese instante, daremos por supuesto que todas las variables han sido adecuadamente dimensionadas.

Continuando con el programa principal, el siguiente paso es crear el bucle infinito que se encargue de mantener su ejecución. Para ello se empleará una simulación de estructura REPEAT/UNTIL, construida mediante el comando BASIC FOR/NEXT:

```
60 FOR BUCLE=0 TO 1 STEP 0
```

El incremento de la variable BLUCLE es 0, por lo que ésta nunca alcanzará el valor 1 que determinaría la salida del bucle; claro está, excepto que se establezca directamente el valor de BUCLE en 2, como se hará en la subrutina de

salida del programa. La penúltima línea del bucle principal, como se verá más tarde, debe coincidir con la instrucción NEXT BUCLE, seguida de la orden END que evite la ejecución imprevista de las siguientes subrutinas. De esta forma se evita además el empleo del comando GOTO, que complicaría el seguimiento del código.

Acto seguido hay que visualizar en la pantalla el menú de opciones, tarea encomendada a las líneas 70 a la 140.

Una vez que se tiene en pantalla el menú de opciones, hay que solicitar al usuario que elija la opción que desee. Puesto que el BASIC de ATARI no posee el comando INKEY$, la captación de la respuesta debe realizarse con el comando equivalente GET. Este necesita la apertura del teclado como dispositivo de entrada de datos, lo que se consigue mediante una instrucción OPEN:

```
150 OPEN # 1, 4, 0, "K:"
```

En donde el 4 denota una operación de entrada de datos y "K:" es el código de dispositivo correspondiente al teclado.

A continuación hay que imprimir en pantalla la petición de pulsar una tecla. Como quiera que el menú ofrece seis opciones, éstas se elegirán mediante un número comprendido de 1 al 6. Desde luego, el programa debe comprobar que la elección es correcta, desechando la pulsación de teclas que no sean dígitos del 1 al 6. El comando GET obtiene el código ASCII de la tecla pulsada; código que debe estar comprendido entre el 49 y el 54 (ambos inclusive) que corresponden a los dígitos del 1 al 6. Todo ello se codifica en las líneas de programa que van a la 160 a la 190.

Cabe observar que se ha simulado de nuevo una estructura REPEAT/UNTIL en la línea 180, la cual establece la salida de este bucle asignando a la variable L el valor 2 si la tecla pulsada es correcta. Si la tecla pulsada no es correcta, se volverá a ejecutar el bucle solicitando la pulsación de una nueva tecla.

Una vez detectada la opción que desea el usuario, sólo habrá que utilizar una instrucción de ON/GOSUB para ejecutar la subrutina correspondiente. La variable selectora ON/GOSUB será T. Al código ASCII de la tecla pulsada habrá que restar el valor 48, para obtener así un número del 1 al 6 que permita la bifurcación a las subrutinas indicadas detrás de la palabra GOSUB. El bucle principal concluirá, por tanto, con las líneas:

```
200 ON T—48 GOSUB 350, 800, 900, 950, 1040, 1110
210 NEXT BUCLE
220 END
```

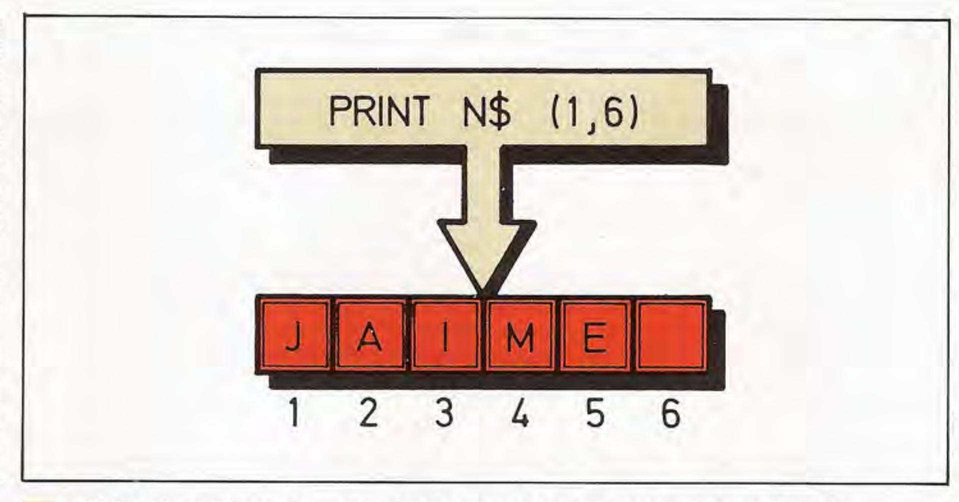
## Adquisición de los datos

Para seguir un orden lógico —el cual no coincide con el indicado en el menú de opciones— hay que estudiar primero la subrutina de introducción en la agenda de un nuevo registro. Es obvio que sin ejecutar esta opción al principio, difícilmente se podrá buscar, modificar o eliminar algún registro. La subrutina para introducir un nuevo registro empezará inicializando las variable oportunas; ésta es, precisamente la misión de las líneas 800 a la 830.

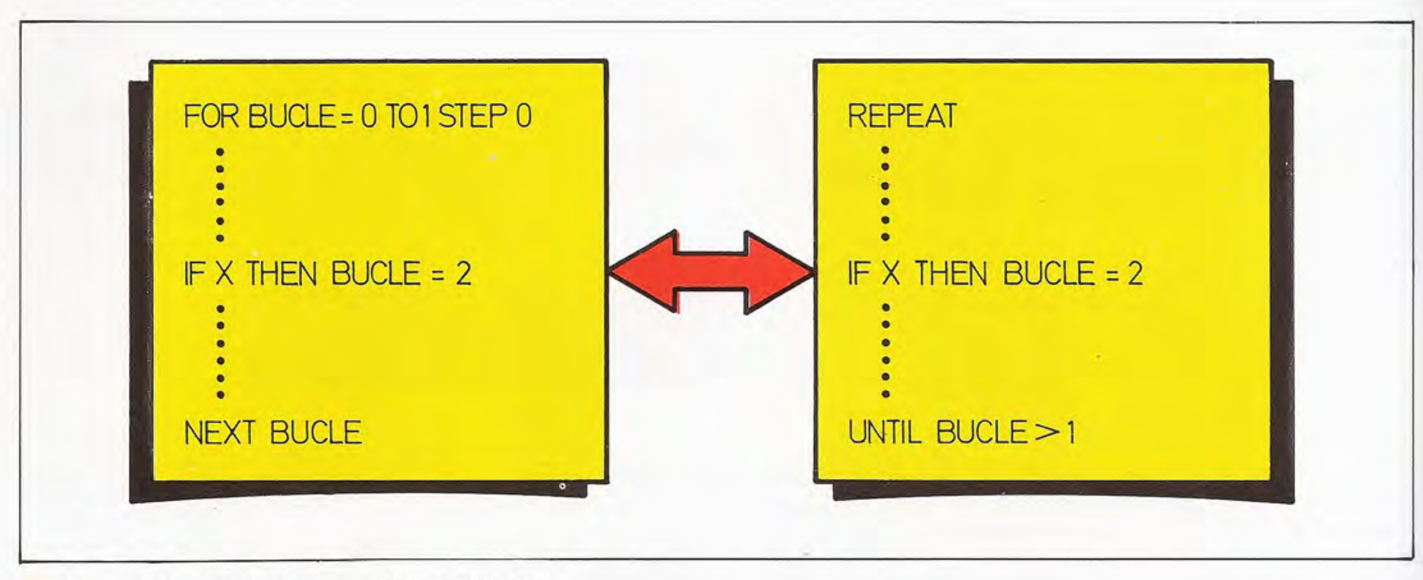
Cada una de las variables utilizadas almacenará temporalmente la información de cada uno de los campos. Así, NOMBRES$ contendrá el nombre completo correspondiente a este nuevo registro, CALLE$ la dirección, CIUDAD$ la localidad de residencia, PROV$ la provincia y TEL$ el número de teléfono. Todas ellas se inicializan con el valor B$, que es una cadena constante de 40 caracteres blancos. El motivo es que otras funciones de la agenda utilizarán también estas variables y pueden contener otras informaciones no deseadas.

La línea 820 llama a la subrutina de la línea 670, que se encarga de la introducción por parte del usuario, mediante el teclado, de los datos del nuevo registro. Dicha subrutina será también utilizada por la función de modificación de registros.

Abandonemos pues, momentáneamente, la subrutina de introducción de nuevo registro, para estudiar esta última (líneas de la 670 a la 790). Resulta de muy fácil asimilación, ya que se limita a introducir mediante comandos INPUT los datos requeridos, para, a continuación, pedir confirmación al usuario de que los datos son correctos. De ser así, se vuelve a interrumpir la estructura REPEAT/UNTIL, nuevamente simulada (línea 77º) mediante el FOR de la línea 700. La entrada de datos con las intrucciones INPUT se realiza después de haber ejecutado la subrutina de la línea 610, la cual imprime en pantalla las cabeceras y valores actuales de los diferentes campos de un registro de la agenda; éstos serán utilizados después como referencia del dato cuya entrada se está solicitando con el comando INPUT. Por este motivo se vuelven a inicializar en blanco las variables de los diferentes campos, para la función de modificación de registros que se verá más adelante. Todo ello puede parecer un tanto complicado; si bien, hay que con-




*El fraccionamiento de una cadena de caracteres se realiza indicando entre paréntesis el principio y el fin de la porción a obtener. Esta técnica equivale a la puesta en práctica por el comando MID$*




*Simulación de una estructura REPEAT/UNTIL por medio del comando FOR/NEXT.*

siderar que de esta forma se evita la necesidad de escribir varias veces las mismas líneas en distintas zonas del programa.

Continuando con la subrutina de entrada de un nuevo registro, la cual se «aparcó» momentáneamente, cabe señalar que en su línea 830 se efectúa una llamada a una nueva subrutina, localizada ésta entre las líneas de programa 480 y 530. Su misión es la de transformar, carácter a carácter, el contenido de la variable NOMBRE$, de forma que quede en mayúsculas (línea 510) y desprovisto de todo carácter distinto de una letra del abecedario (línea 520).

El objeto de esta transformación no es otro que crear un índice estandarizado que permita acceder a través del mismo a los restantes campos de información de la agenda. Para no dilatar en exceso el programa, sólo se va a codificar la creación de un índice para la localización de registros por su nombre; aunque, bien es cierto, que también se podría realizar de la misma forma para acceder al registro por el campo de dirección, de ciudad o de teléfono, sin más que crear un índice para cada uno de estos campos. Así pues, en la variable NOM$ se obtendrá el nombre en mayúsculas y sin signos extraños del registro en curso. Este valor será introducido en la cadena N$, que constituirá el índice de los nombres de los registros que estén almacenados en el disco. De la actualización de este índice con el nuevo nombre se encarga la siguiente línea:

```
840 N$(P*40-39, P*40)=NOM$: N$(4001)=" "
```

La variable N$ se supone previamente dimensionada con 4001 caracteres. Esto es así porque el BASIC del ATARI no admite matrices multidimensionales de cadenas de caracteres; de ahí que sea necesario simularlas mediante una única cadena que será artificialmente «cortada» en los trozos oportunos.

Como la longitud máxima del nombre es de 40 caracteres (longitud adoptada para ese campo), y se ha previsto una capacidad máxima de 100 registros en la agenda (cantidad que puede ser modificada a voluntad), esto hace un total de 40*100=4000 caracteres, que serán necesarios para el índice. Cada subcadena, dentro de la cadena N$, será localizada gracias a la variable P, que contendrá en todo momento el número de posición del siguiente registro libre, con lo que el nombre para el índice del nuevo registro guardado hasta ahora en NOM$, se almacenará en N$(P*40-39, P*40). El hecho de ordenar que el último carácter de N$ sea " ", se debe a que al asignar un valor a una subcadena de N$, la longitud de ésta queda recortada a P*40, lo que podría dar lugar a errores al querer buscar posteriormente una subcadena en N$.

## Creación del archivo de acceso aleatorio

Una vez actualizado el índice, sólo queda almacenar el registro en el archivo del disco; tarea ésta cuya ejecución corresponde a las líneas de la 850 a la 890.

En la línea 850 se abre el archivo AGENDA.DAT en modalidad APPEND (ver figura adjunta), con lo que el nuevo registro se podrá incorporar al final del archivo. Pero, ¿qué ocurre si el programa es la primera vez que se ejecuta y no existe todavía en el disco? Pues que se producirá un error de «archivo no encontrado». El comando TRAP de esta misma línea, hará que al producirse ese error se derive la ejecución del programa a la línea 890, en la que se creará por vez primera el archivo AGENDA.DAT.

Los archivos de acceso aleatorio del ATARI no son diferentes a los de acceso secuencial, como ocurre en otras versiones del BASIC. En éstos se puede obtener la posición exacta del disco en la

que se graba un registro y almacenaría en alguna variable, para, posteriormente, acceder directamente a esa posición. El comando del BASIC-ATARI que se encarga de tal cometido es NOTE, cuyo formato es el siguiente:

NOTE # <canal>, <var. 1>, <var. 2>

El parámetro <canal>es el número identificador de un archivo previamente abierto en las modalidades WRITE, APPEND o UPDATE, mientras que <var. 1>, <var. 2>, son dos variables numéricas.

El funcionamiento de este comando es el siguiente: la unidad de disco (o más bien el D.O.S.) posee un puntero que señala al sector y al byte concreto de dicho sector al que se está accediendo en cada momento. El comando NOTE accede a ese puntero y almacena el número de sector en la primera variable específica (<var. 1>) y el número del byte de ese sector en <var. 2>. Una vez leídos estos valores, no queda más que grabar la información mediante in PRINT# o un PUT#. Como se puede ver en el programa de agenda, la línea 860 obtiene con NOTE la posición del disco en la que se va a grabar la información y después la almacena en la matriz numérica bidimensional INDICE(,), para saber dónde encontrarla cuando sea necesario. Seguidamente «imprime» en esa posición los diferentes campos del registro y cierra el archivo, con lo que la información queda ya a buen recaudo; tras ello actualiza la variable P para el siguiente acceso y se devuelve al ordenador el control de errores mediante TRAP 40000. Con ello finaliza la subrutina de introducción de un nuevo registro y se regresa al programa principal para presentar en pantalla el menú de opciones.

## En busca del registro perdido

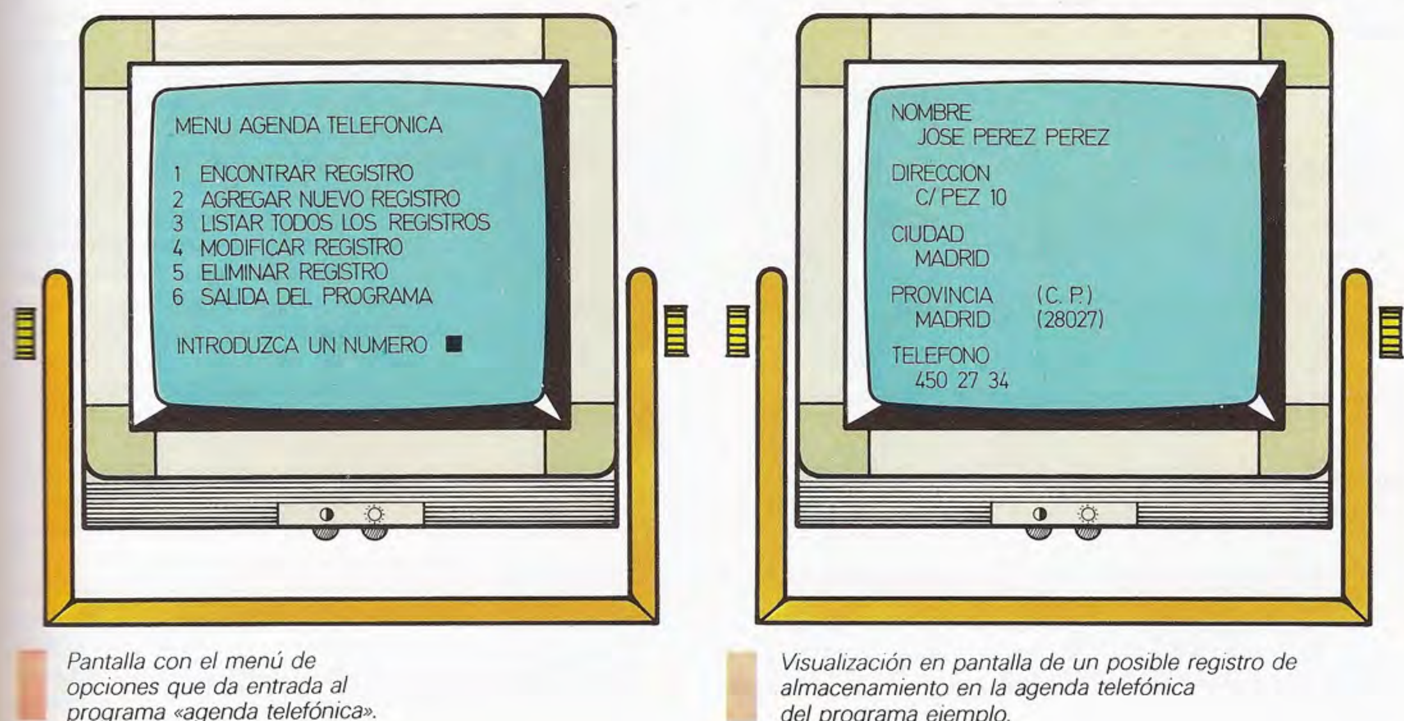
Para recuperar un registro grabado en disco se pulsará la tecla 1, con lo que el programa saltará a la subrutina de la línea 350, la cual se extiende hasta la 430.

En primer lugar se pierde el nombre que se desea buscar, almacenándolo mediante un INPUT en la variable NOMBRE$. La variable POS empieza con valor 1, e irá recorriendo uno a uno todos los caracteres ocupados de N$, hasta hallar una equivalencia entre el nombre a buscar (almacenado en NOM$) y una subcadena de N$; dicha misión se encomienda a la subrutina llamada en la línea 380 y que ocupa las líneas comprendidas entre la 440 y 470.

Una vez localiza el nombre en cuestión, se halla el valor de la variable CLAVE. Este coincidirá con el número de registro en el que está guardada la información solicitada. Acto seguido se actualiza POS, por si ese nombre no es el deseado y hay que seguir buscando. Esto es así debido a que dos registros pueden tener el mismo nombre, por lo que al solicitar ese nombre obtendrán los dos.

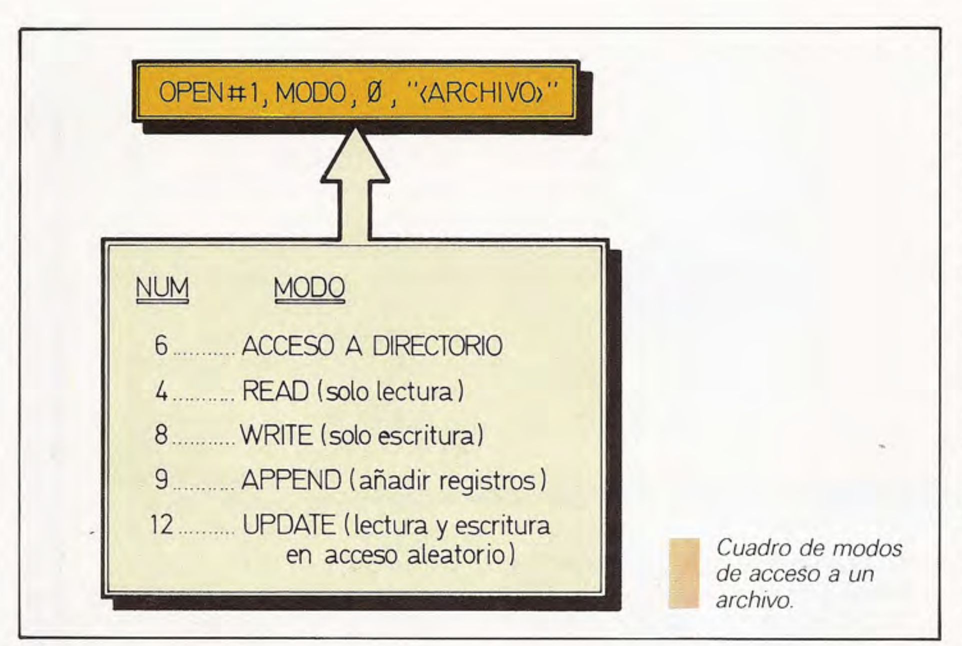
La subrutina que nos ocupa devolverá el control al punto de llamada cuando encuentre el nombre buscado, o bien cuando no encuentre ese nombre en todo el índice; es este último caso, retornará con el valor de CLAVE 0. De acuerdo al valor devuelto como clave, la subrutina de búsqueda de registro imprimirá el mensaje de «no encontrado» y volverá al programa principal cuando sea CLAVE=0. No obstante, si el contenido de CLAVE es distinto de 0, pasará a leer ese registro del disco mediante la subrutina llamada en la línea 400, la cual se extiende desde 540 hasta la línea de programa 600.

Esta subrutina abre el archivo AGENDA.DAT que contiene la información de la agenda, en modo READ. Para acceder de forma aleatoria a la información




*Pantalla con el menú de opciones que da entrada al programa «agenda telefónica».*

*Visualización en pantalla de un posible registro de almacenamiento en la agenda telefónica del programa ejemplo.*




*Cuadro de modos de acceso a un archivo.*

guardada en él, será necesario utilizar el comando del BASIC-ATARI: POINT#, complementario del NOTE#, y cuyo formato es el siguiente:

POINT # <canal>, <var. 1>, <var. 2>

Este realiza la operación contraria a NOTE#, de tal forma que permite situar el puntero de la unidad de disco en la posición que se desee; posición que se indicará mediante el contenido de las variables <var. 1> y <var. 2>, para el sector y el byte de dicho sector, respectivamente. Como se recordará, esta posición estaba almacenada en la variable INDICE(,), la cual será leída mediante el valor de CLAVE como se muestra en la línea 580. Una vez posicionado el puntero, sólo queda ejecutar las correspondientes instrucciones INPUT# para leer la información del disco. Tras presentar la información leída en la pantalla mediante la subrutina 610, se devuelve el control al programa principal.

## POINT #

Posiciona el puntero del disco en el registro de acceso aleatorio al que se desea acceder.

*Formato:* <n. 1> POINT#<canal>, <var. 1>, <var. 2>

*Ejemplos:*
```
30 POINT #3,IND(1), IND(2)
50 POINT #3, SECTOR, BYTE : INPUT #3; A$
```

## NOTE #

Obtiene la posición del puntero de la unidad de disco en la que se va a efectuar una grabación de un registro de acceso aleatorio.

*Formato:* <n. 1>NOTE #<canal>, <var. 1>, <var. 2>

*Ejemplos:*
```
10 NOTE #2, SECTOR, BYTE
20 NOTE #1, IND(1), IND(2) : PRINT #2; A$
```

Las restantes opciones utilizarán las mismas subrutinas que ya se han explicado hasta aquí, por lo que resultará muy sencillo comprender su funcionamiento. Por ejemplo, para listar todos los registros, sólo habrá que crear un blucle que recorra todos los registros uno a uno, e ir leyendo la información del disco y presentándola en pantalla. Esta subrutina es la comprendida entre las líneas 900 y 940.

Tras presentar en pantalla un registro, se esperará a que el usuario pulse una tecla cualquiera para visualizar el siguiente.

La subrutina de modificar un registro coincidirá también con una combinación de las subrutinas ya estudiadas. Concretamente, la zona básica de la misma se sitúa entre las líneas de programa 950 y 1030. En ella se pide en primer lugar el nombre del registro a modificar; éste es buscado de forma similar a como se hacía en la opción de encontrar un registro. Después se visualiza en pantalla y mediante instrucciones INPUT se modifican los campos necesarios. Una vez modificado, se deposita el contenido en el archivo empleando el comando POINT# y los PRINT#; la única salvedad es que en este caso el archivo se abre en modalidad UPDATE. Previamente se habrá actualizado el índice N$, con lo que se da por terminada esta subrutina.

La subrutina de eliminar un registro reduce su actuación a borrar a este del índice (variables INDICE() y N$), con lo que ya no será accesible. Dicha eliminación se opera desplazando hacia atrás, y uno a uno, los valores que lo siguen en las variables índice; si bien, no se elimina el archivo del disco, por lo que será fácilmente recuperable en el caso de que se haya ordenado su descarte por equivocación. La subrutina al efecto es la situada entre las líneas 1040 y 1100.

Finalmente, la opción de salida del programa se encargará de almacenar en el disco el archivo secuencial "INDICE.DAT", las variables P, INDICE() y N$, ya que hasta ahora éstas sólo residían en la memoria principal del ordenador y son esenciales para recuperar la información de la agenda. Por este motivo, no se debe ordenar la ruptura del programa en ningún punto del mismo; siempre hay que abandonar el programa ejecutando la opción 6 de salida. De

otra forma se podría perder toda la información actualizada.

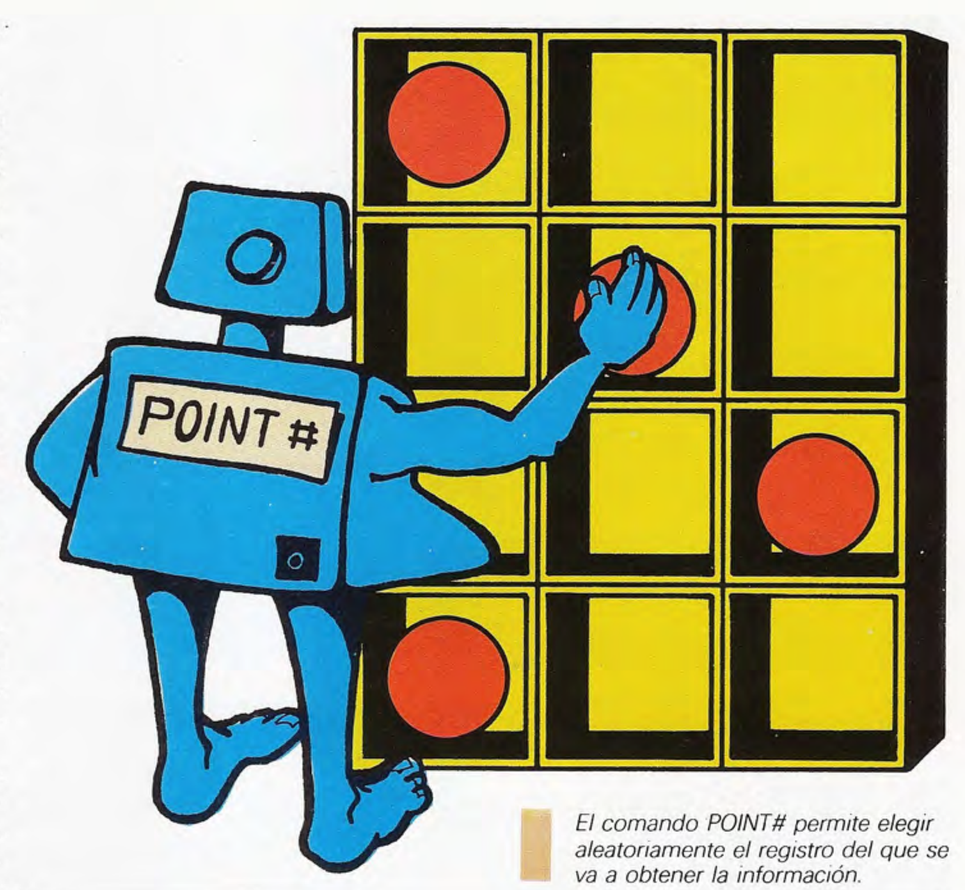
La grabación de los índices para el acceso se podría haber realizado tras cada actualización del archivo AGENDA.DAT, bien por añadir un nuevo registro, o por modificar o eliminar alguno ya existente; no obstante, ello haría que el manejo de la agenda resultara más lento y tedioso. La subrutina de salida está emplazada entre las líneas 1110 y 1160 del programa.

Como puede observarse, una vez grabadas las variables de índice, se ejecuta un GRAPHICS 0 que inicializa el sistema, y se asigna a BUCLE el valor 2 que concluirá la ejecución de la estructura REPEAT/UNTIL simulada en el programa principal; con ello se alcanzará la instrucción END.

A estas alturas sólo queda por comentar la subrutina de inicialización (líneas 230 a la 340), relegada hasta el final para su mejor comprensión. Ahora se conocen ya las distintas variables utilizadas en el programa.

En ella se empieza dimensionando las distintas variables utilizadas en el programa, de acuerdo con la dimensión requerida y prefijada en las líneas precedentes. La línea 250 de la mencionada subrutina pone en práctica un artificio, específico de los ordenadores de ATARI, que permite una rápida inicialización de cadenas de caracteres con un valor concreto (en cada caso caracteres blancos para B$ y N$).

A continuación, se obtienen del archivo AGENDA.DAT los valores para las variables P, INDICE( ) y N$, que permitirán el acceso aleatorio al archivo AGENDA.DAT que memoriza la información de la agenda. En el caso de tratarse de la primera ejecución del programa —aún no existe en el disco el archivo AGENDA.DAT—, se producirá un error de «fichero no encontrado»; error que es manipulado mediante el comando TRAP, de forma similar a como se explicó para la subrutina de creación del archivo AGENDA.DAT. En este caso no se creará el archivo INDICE.DAT, sino que se asignará a la variable P el valor 1, que revelará que no existen todavía registros en la agenda. De inmediato se retornará al programa principal para comenzar el bucle que visualiza el menú y solicita al usuario que elija una opción.




*El comando POINT# permite elegir aleatoriamente el registro del que se va a obtener la información.*




*El comando NOTE# permite obtener el valor de la posición del disco (sector y byte) sobre la que se va a grabar la siguiente información.*

**AGENDA ELECTRONICA**

```
10 REM AGENDA ELECTRONICA
20 PRINT CHR$(125) : POKE 752, 1 : REM BORRAR PANTALLA Y ELIMINAR
   CURSOR
30 POSITION 10, 9 : PRINT "AGENDA TELEFONICA"
40 GOSUB 230 : REM INICIALIZAR VARIABLES
50 POKE 752, 0 : REM HACER APARECER CURSOR
60 FOR BUCLE=0 TO 1 STEP 0
70 PRINT CHR$(125) : REM BORRAR PANTALLA
80 POSITION 8, 1 : PRINT "MENU AGENDA TELEFONICA"
90 PRINT : PRINT : PRINT "1. ENCONTRAR REGISTRO"
100 PRINT : PRINT "2. AGREGAR NUEVO REGISTRO"
110 PRINT : PRINT "3. LISTAR TODOS LOS REGISTROS"
120 PRINT : PRINT "4. MODIFICAR REGISTRO"
130 PRINT : PRINT "5. ELIMINAR REGISTRO"
140 PRINT : PRINT "6. SALIDA DEL PROGRAMA"
150 OPEN #1, 4, 0, "K:"
160 FOR L=0 TO 1 STEP 0
170 POSITION 7, 16 : PRINT "SELECCIONE UN NUMERO  "; : GET #1, T :
    PRINT CHR$(T)
180 IF T>48 AND T<55 THEN L=2
190 NEXT L : CLOSE #1
200 ON T-48 GOSUB 350, 800, 900, 950, 1040, 1110
210 NEXT BUCLE
220 END
230 REM SUBRUTINA DE INICIALIZACION
240 DIM NOMBRE$(40), T$(1), NOM$(40), CALLE$(40), CIUDAD$(40),
    PROV$(40), TEL$(40), B$(40)
250 DIM N$(4001), INDICE(100, 1) : N$=" " : N$(4001)=" " : N$(2)=N$ :
    B$=" " : B$(40)=" " : B$(2)=B$
260 TRAP 340 : OPEN #3, 4, 0, "D : INDICE.DAT"
270 INPUT #3; P
280 FOR I=1 TO P-1
290 INPUT #3; NOMBRE$ : N$(I*40-39)=NOMBRE$
300 NEXT I : N$(4001)=" "
310 FOR I=1 TO P-1
320 INPUT #3; T : INDICE(I, 0)=T : INPUT #3; T : INDICE(I, 1)=T
330 NEXT I : CLOSE #3 : TRAP 40000 : RETURN
340 CLOSE #3 : P=1 : TRAP 40000 : RETURN
350 REM SUBRUTINA BUSCAR REGISTRO
360 PRINT : PRINT "INTRODUZCA NOMBRE A BUSCAR" : INPUT NOMBRE$
370 GOSUB 480 : POS=1
380 FOR L=0 TO 1 STEP 0 : GOSUB 440 : REM BUSCAR NOMBRE EN INDICE
390 IF CLAVE=0 THEN PRINT : PRINT "NOMBRE NO ENCONTRADO"; : L=2 :
    FOR K=1 TO 200 : NEXT K
400 IF CLAVE<>0 THEN GOSUB 540 : PRINT : PRINT "SIGO BUSCANDO ?
    (S/N) "; : OPEN #1, 4, 0, "K:" : GET #1, T : T$=CHR$(T) : CLOSE #1
410 IF CLAVE <>0 AND (T$="N" OR T$="n") THEN L=2
420 NEXT L
430 RETURN
440 REM SUBRUTINA BUSCAR NOMBRE
450 CLAVE=0 : FOR I=POS TO (P-1)*40-LEN(NOM$)
460 IF N$(I, I+LEN(NOM$)-1)=NOM$ THEN CLAVE=INT(I/40)+1 : POS=I+1 :
    I=(P-1)*40-LEN(NOM$)
470 NEXT I : RETURN
480 REM SUBRUTINA MODIFICAR NOMBRE PARA INDICE
490 NOM$=" " : N=1 : FOR L=1 TO LEN(NOMBRE$)
500 T$=NOMBRE(L, L) : T=ASC(T$)
510 IF T>=97 THEN T=T-32
520 IF T>64 AND T<91 THEN NOM$(N, N)=CHR$(T) : N=N+1
530 NEXT L : RETURN
540 REM SUBRUTINA LECTURA DE DATOS
550 IF P=1 THEN PRINT : PRINT "  AGENDA VACIA" : RETURN
560 PRINT CHR$(125); : REM LIMPIAR PANTALLA
570 OPEN #2, 4, 0, "D : AGENDA.DAT"
580 POINT #2, INDICE(CLAVE, 0), INDICE(CLAVE, 1)
590 INPUT #2; NOMBRE$ : INPUT #2; CALLE$ : INPUT #2; CIUDAD$
600 INPUT #2; PROV$ : INPUT #2; TEL$ : CLOSE #2 : GOSUB 610 : RETURN
610 REM SUBRUTINA IMPRIMIR REGISTRO EN PANTALLA
620 PRINT CHR$(125); "NOMBRE" : PRINT "  "; NOMBRE$ : PRINT "DIRECCION"
 : PRINT "  "; CALLE$
630 PRINT "CIUDAD" : PRINT "  "; CIUDAD$
640 PRINT "PROVINCIA (C. P.)" : PRINT "  "; PROV$
650 PRINT "TELEFONO" : PRINT "  "; TEL$
660 RETURN
670 REM SUBRUTINA INTRODUCIR DATOS POR TECLADO
680 PRINT CHR$(125)
690 GOSUB 610 : NOMBRE$=B$ : CALLE$=B$ : CIUDAD$=B$ : PROV$=B$ :
    TEL$=B$
700 FOR L=0 TO 1 STEP 0
710 POSITION 2, 1 : INPUT NOMBRE$ : NOMBRE$(40)=" "
720 PRINT : PRINT : INPUT CALLE$ : CALLE$(40)=" "
730 PRINT : PRINT : INPUT CIUDAD$ : CIUDAD$(40)=" "
740 PRINT : PRINT : INPUT PROV$ : PROV$(40)=" "
750 PRINT : PRINT : INPUT TEL$ : TEL$(40)=" "
760 PRINT : PRINT : PRINT "SON CORRECTOS LOS DATOS ? (SI/NO) "; :
    OPEN # 1, 4, 0, "K:" : GET #1, T : CLOSE #1 : T$=CHR$(T)
770 IF T$="S" OR T$="s" THEN L=2
780 NEXT L
790 RETURN
800 REM SUBRUTINA INTRODUCIR NUEVO REGISTRO
810 NOMBRE$=B$ : CALLE$=B$ : CIUDAD$=B$ : PROV$=B$ : TEL$=B$
820 GOSUB 670 : REM ENTRADA POR TECLADO
830 GOSUB 480 : REM MODIFICAR NOMBRE PARA INDICE
840 N$(P*40-39, P*40)=NOM$ : N$(4001)=" "
850 TRAP 890 : OPEN #2, 9, 0, "D : AGENDA.DAT"
860 NOTE #2, SECT, BYTE : INDICE(P, 0)=SECT : INDICE(P, 1)=BYTE
870 PRINT #2; NOMBRE$ : PRINT #2; CALLE$ : PRINT #2; CIUDAD$ :
    PRINT #2; PROV$; PRINT #2; TEL$
880 CLOSE #2 : P=P+1 : TRAP 40000 : RETURN
890 TRAP 40000 : CLOSE #2 : OPEN #2, 8, 0, "D : AGENDA.DAT" : GOTO 860 :
    REM CREAR ARCHIVO SI NO EXISTIA YA
900 REM SUBRUTINA LISTAR REGISTROS
910 FOR IND=1 TO P-1
920 CLAVE=IND : GOSUB 540
930 PRINT : PRINT "  PULSE UNA TECLA PARA SEGUIR  "; : OPEN #1, 4, 0,
 "K:" : GET #1, T : CLOSE #1
940 NEXT IND : RETURN
950 REM SUBRUTINA MODIFICAR REGISTRO
960 PRINT : PRINT "INTRODUZCA NOMBRE A MODIFICAR" : INPUT NOMBRE$
970 GOSUB 370 : IF CLAVE=0 THEN RETURN : REM BUSCAR REGISTRO A
    MODIFICAR
980 GOSUB 690 : REM ENTRADA DE DATOS POR TECLADO
990 GOSUB 480 : N$(CLAVE*40-39, CLAVE*40)=NOM$ : N$(4001)=" "
1000 OPEN #2, 12, 0, "D : AGENDA.DAT"
1010 POINT #2, INDICE(CLAVE, 0), INDICE(CLAVE, 1)
1020 PRINT #2; NOMBRE$ : PRINT #2; CALLE$ : PRINT #2; CIUDAD$ :
     PRINT #2; PROV$ : PRINT #2; TEL$
1030 CLOSE #2 : RETURN
1040 REM SUBRUTINA ELIMINAR REGISTRO
1050 PRINT : PRINT "INTRODUZCA NOMBRE A ELIMINAR" : INPUT NOMBRE$
1060 GOSUB 370 : IF CLAVE=0 THEN RETURN : REM BUSCAR REGISTRO A
     MODIFICAR
1070 FOR L=CLAVE TO P-1
1080 N$(L*40-39, L*40)=N$((L+1)*40-39, (L+1)*40) : N$(4001)=" "
1090 INDICE(L, 0)=INDICE(L+1, 0) : INDICE(L, 1)=INDICE(L+1, 1)
1100 NEXT L : P=P-1 : RETURN
1110 REM SALIDA DEL PROGRAMA
1120 IF P=1 THEN GRAPHICS 0 : BUCLE=2 : RETURN
1130 OPEN #3, 8, 0, "D : INDICE.DAT"
1140 PRINT #3; P : FOR L=1 TO P-1 : NOMBRE$=N$(L*40-39, L*40) :
     PRINT #3; NOMBRE$ : NEXT L
1150 FOR L=1 TO P-1 : PRINT #3; INDICE(L, 0) : PRINT #3; INDICE(L,1) :
     NEXT L
1160 CLOSE #3 : GRAPHICS 0 : BUCLE=2 : RETURN
```

Este programa pretende tan sólo ilustrar claramente el manejo de archivos de acceso aleatorio, por lo que en ciertos aspectos está simplificado. Con su estructura actual —reflejada en el listado adjunto—, resulta ya perfectamente útil en la práctica; no obstante, su máximo rendimiento se obtendrá al introducir el usuario sus propias mejoras, adaptándolo a sus necesidades específicas. Estas modificaciones pueden concretarse, por ejemplo, en permitir el acceso por otros campos diferentes al de nombre, o ampliar la capacidad de la agenda a más de 100 registros.

# Los archivos del Commodore 64

## Tratamiento de archivos en la unidad de disco 1541

La unidad de disco básica de la familia Commodore (apellidada 1541) es un periférico inteligente. En efecto, posee su propio microprocesador de 8 bits (un 6502), rodeado de 2 Kbytes de memoria RAM y de una zona de memoria ROM en la que está almacenado el sistema operativo de disco (DOS).

Esta «inteligencia» facilita el tratamiento o manipulación de los archivos almacenados en el disquete, independientemente de la tarea que en cada instante esté ejecutando el microprocesador que dirige las actividades de la unidad central. Así, por ejemplo, la unidad de disco puede dar salida por sí misma a algunos archivos; enviando la información hacia la impresora para que ésta imprima el contenido de los mismos en papel. Mientras tanto, el ordenador puede ejecutar cualquier otra tarea, puesto que no se ve en la necesidad de atender al proceso que está en curso entre la unidad de disco y la impresora.

Las operaciones sobre los archivos almacenados en disco se desencadenan a través de «órdenes» BASIC o por medio de comandos del sistema operativo; por supuesto, esgrimiento los números de archivos lógicos adecuados y los números de dispositivos periféricos correspondientes, cuyos formatos se estudiarán en los próximos apartados. Hay que tener en cuenta (y de ahí el motivo del presente capítulo) que las órdenes a utilizar en el caso que nos ocupa difieren de las establecidas para otros intérpretes BASIC.



*La incorporación de la unidad de disco al equipo Commodore 64 potencia las posibilidades de este ordenador, capacitándolo para ejecutar múltiples aplicaciones de gestión en las que resulta imperativo el trabajo con archivos de acceso aleatorio.*

### El formato de los discos

El sistema operativo de disco de Commodore ofrece una primera ventaja significativa: este no debe cargarse a partir de disquete, puesto que reside en la ROM incluida en la circuitería de la propia unidad de disco. Ello significa que las funciones del DOS están disponibles desde el preciso momento en el que entra en funcionamiento la unidad de disco conectada al ordenador.

Por otro lado, el DOS maneja un amplio conjunto de órdenes destinadas a que el usuario pueda codificar complicados programas que manipulen tanto los archivos de programas, como los archivos de datos del usuario.

Pero vayamos al principio. Como es natural en este tipo de soportes, antes de empezar el trabajo con un disco es preciso formatearlo en pistas y sectores. El formato de los disquetes que proporciona el DOS consta de 35 pistas, divididas en sectores. Dado que las pistas más internas tienen una longitud inferior a la de las pistas más externas, el DOS las divide en distinto número de sectores. Así, esta división va desde los 17 sectores para la pista más interna, hasta 21 sectores para la más externa. Por su parte cada sector contiene un bloque de datos de 256 bytes, más una serie de bytes de control necesarios para el correcto funcionamiento del sistema. De esta forma se obtiene un espacio máximo de 170 Kbytes libres por disco flexible de 5 y 1/4 pulgadas.

El formateado del disquete se completa con un mapa de disponibilidad de bloques (BAM=Block Availability Map) y un directorio de los archivos del disco. Las referidas tablas sirven para administrar la distribución de los datos en el disco. Ambas están almacenadas en la pista 18 del disquete. Más concretamente, el BAM se encuentra situado en el sector 0 de dicha pista y está constituido por 144 bytes que señalan qué bloques están libres para el almacenamiento de datos y cuáles no. El directorio del disquete se sitúa a partir del sector 1, y es una lista que puede incluir hasta 144 nombres de archivos asociados a información relativa al tipo de archivo y al número de bloques que lo constituyen.

El BAM y el directorio se van actualizando automáticamente a medida que se introducen datos en el disquete, aunque también es posible acceder a ellos directamente, como se verá más adelante.

### Los archivos de programas

Los programas BASIC se almacenan en el disquete como archivos binarios. Para ello se dispone de las tradicionales órdenes BASIC LOAD y SAVE, análogas a la utilizadas para el almacenamiento de datos en casete, aunque con ciertas diferencias en su formulación:

LOAD <nombre$>, <dispositivo> [, <comando>]

Donde el nombre del programa (<nombre$>) es una cadena de carac-






*La unidad de disco 1541 es un periférico de la gama Commodore compatible con los populares VIC-20 y Commodore 64.*

teres válida, es decir: un nombre encerrado entre comillas o el contenido de la variable alfanumérica especificada. El <dispositivo> es un número entero que identifica al periférico al que se quiere acceder. Inicialmente, este número está fijado por hardware, de forma que a la unidad de disco le corresponde el número 8; si bien, es posible modificar este número, ya sea por software o por hardware, para así poder conectar simultáneamente más de una unidad de disco. El <comando> es otro número entero cuyo uso es opcional; si se omite o se hace igual a 0, el programa se carga normalmente, mientras que si se le otorga el valor 1 se cargará en las mismas posiciones de memoria que ocupaba al ser traspasado al disquete. Veamos algunos ejemplos:

```
LOAD "Programa prueba", 8
LOAD NOMBRE$, 8, 1
LOAD N$, D, C
```

El comando LOAD se utiliza también para «leer» el directorio del disquete. Este recibe un tratamiento por parte del DOS como si se tratara de un programa BASIC, de forma que puede ser «cargado» en la memoria del ordenador y «listado» para examinar su contenido. El nombre asignado a este «programa» es "$", de forma que la orden completa que permite «cargar» el directorio del disquete será:

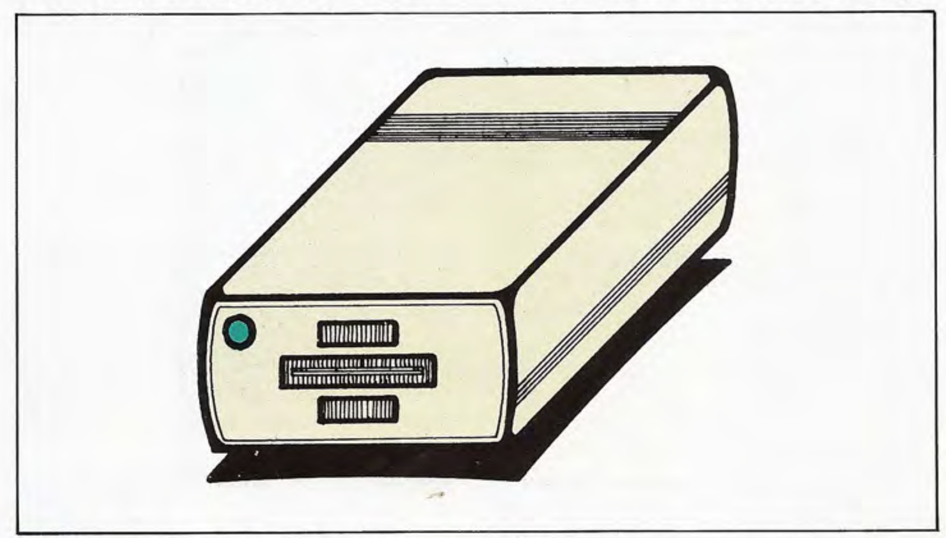
```
LOAD "$", 8
```

Y una vez cargado no queda más que ejecutar la orden LIST para que aparezca el directorio en la pantalla. Este listado contiene la siguiente información:

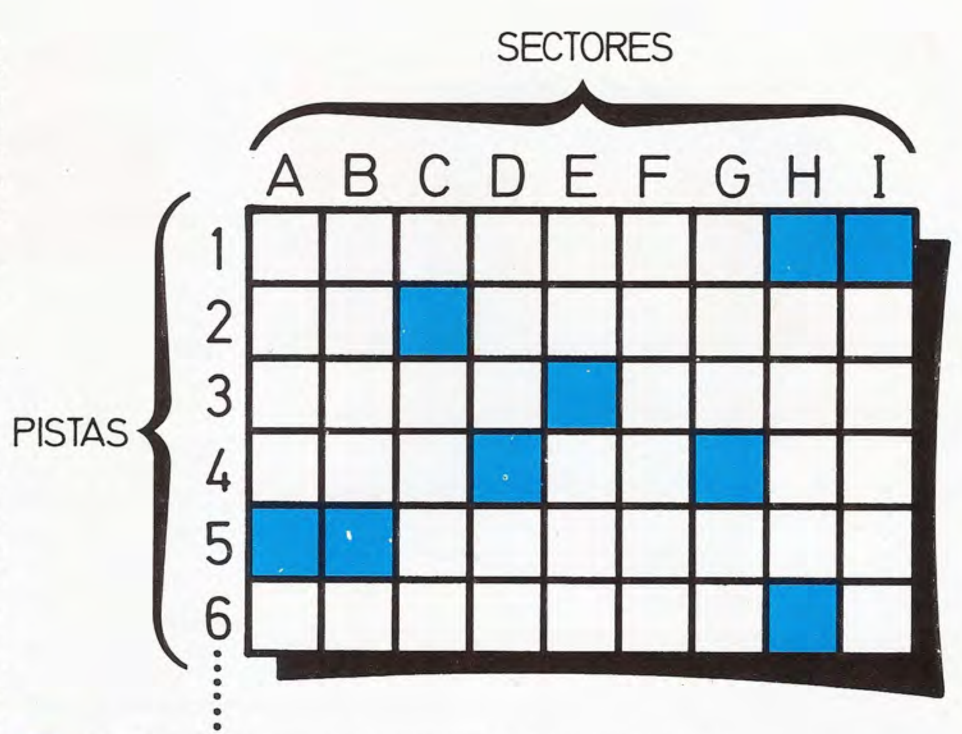
- Nombre del disquete.
- Identificador (de dos caracteres) del disquete.
- Hasta 144 nombres de archivos.
- Tipo de cada archivo.
- Longitud en bloques de cada archivo.
- Número de bloques libres disponibles.

Los nombres de los programas almacenados en el disquete, admiten también los clásicos «comodines» o «wildcards» («*» y «?»). Con ello es posible que una misma orden haga referencia a múltiples archivos que compartan tan sólo alguna característica en sus respectivos nombres.

Hay que tener en cuenta que, debido al tratamiento que se le da al directorio con este método, su carga «destruye» cualquier otro programa BASIC residen-



*La unidad de disco 1541 de Commodore es un periférico «inteligente», que incorpora su propio microprocesador del tipo 6502.*




*El BAM lleva la cuenta de los sectores del disco que han sido ocupados por datos y de los que permanecen disponibles.*

te en ese preciso momento en la memoria del ordenador, con lo que se pierde esa información. En todo caso, también es posible leer el directorio sin perder la información previamente almacenada en memoria; ello supone leer el archivo "$" por medio del comando GET#, como si se tratara un archivo secuencial.

El formato del comando SAVE es totalmente análogo al de la LOAD ya descrita:

SAVE <nombre$>, <dispositivo> [,<comando>]

Por lo que respecta a la actuación de este comando, hay que considerar que se producirá un error en la unidad de disco cuando se intente almacenar un archivo con un nombre ya existente en el disco activo. En tal situación, será necesario leer el «canal de errores», borrar el archivo antiguo, y almacenar entonces el nuevo. Este inconveniente se puede evitar sin más que indicar en el comando SAVE lo siguiente:

SAVE "@0 : "+NOMBRE$, 8

Con lo que el nuevo archivo reemplazará automáticamente al antiguo, sin necesidad de recurrir a otros comandos adicionales.

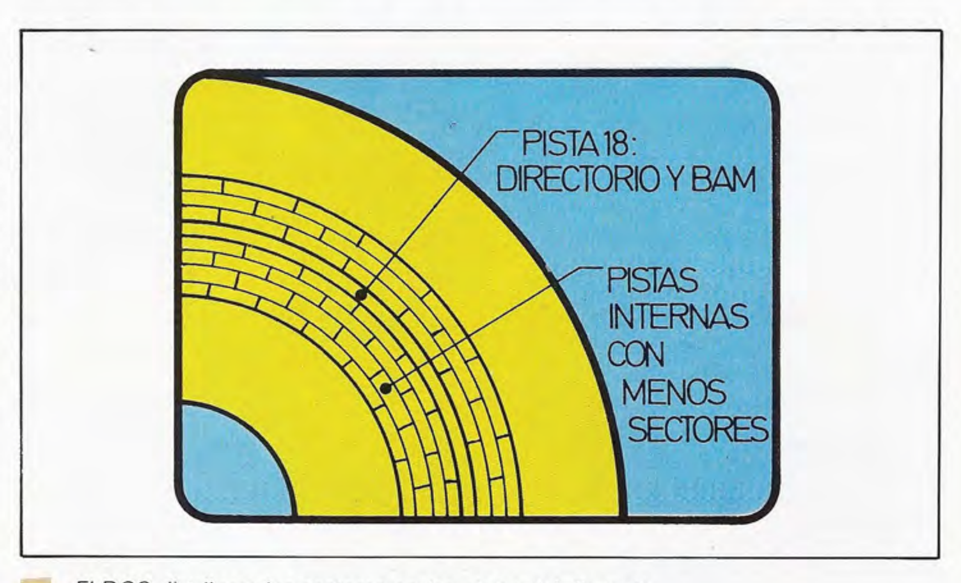
## Los comandos del disco

El cometido y principales aplicaciones de los comandos OPEN y PRINT# del lenguaje BASIC, han sido ya descritos en un capítulo anterior de esta obra. Respecto a su funcionamiento con la unidad de discos de Commodore, cabe precisar que su uso es semejante al definido como caso general. La novedad de Commodore consiste en que también pueden ser utilizados para abrir un canal para comandos de disco que controlen los intercambios de información entre el ordenador y la referida unidad.

El formato para el comando OPEN es:

OPEN <archivo>, <dispositivo>, <canal>, <texto$>

Donde: <archivo> es un número entero comprendido entre 1 y 255 que identificará durante el resto del programa al archivo al que se quiere acceder. El número del <dispositivo> es, para el caso de la unidad de discos, el 8 y ya ha sido comentado con anterioridad. El <canal> debe ser un número comprendido entre 1 y 15, y se refiere al canal de comunicación entre unidad de disco y ordenador que se desea utilizar. Los números 0 y 1 ya han sido comentados




*El DOS distribuye los sectores en las pistas del disquete, de tal forma que los interiores «tocan» a menos sectores por ser inferior su longitud.*




*El directorio contiene información de diversa índole relativa a los archivos almacenados en disco.*

y se reservan para su uso con LOAD y SAVE. Del 2 al 15, se utilizan para el trasvase de datos a archivos, y el canal 15 es a través del cual se deben enviar los comandos de disco. El <texto$> es una cadena de caracteres válida, que será enviada al archivo abierto, al igual que si se ejecutase un comando PRINT#; ello significa que pueden enviarse comandos a través del canal con el auxilio de la orden OPEN o directamente con PRINT#.

El comando PRINT# funciona análogamente al PRINT convencional, salvo que en vez de enviar el mensaje a la pantalla lo enviará por el canal especificado hacia el archivo previamente abierto. Finalmente, se debe cerrar todo archivo o canal abierto, que ya no sea necesario, mediante la ya conocida orden CLOSE.

Entre los distintos comandos disponibles, quizá el de primera necesidad sea el que se utiliza para formatear o inicializar el disquete, de lo contrario no será posible acceder al disco. Este comando es NEW, cuyo cometido es borrar completamente el soporte magnético, definir en él las marcas de control de los bloques, y crear el directorio y el BAM. Su formulación es la siguiente:

PRINT#15, "NEW:<nombre$> [,<identificador>]"

En donde: <nombre$> es una cadena de caracteres válida que constituye el nombre que se asignará al disco y que aparecerá en la cabecera del directorio. El <identificador> constará de dos caracteres únicamente y su uso es opcional. Sirve para caracterizar a todos los ficheros de ese disquete, y evitar que se cometa algún error al intentar acceder a un archivo de otro disco. Un ejemplo de formulación es el que sigue:

PRINT#15, "NEW:disco de pruebas, LC"

Otro comando de utilidad es COPY, el cual sirve para obtener copias en el mismo disco de un programa o archivo ya residente en el mismo, aunque asignándole un nuevo nombre.

Por ejemplo:

PRINT#15, "COPY:COPIA=ORIGINAL"

Este mismo comando permite también crear un nuevo archivo de tipo secuencial, que sea la concatenación de hasta otros cuatro archivos secuenciales diferentes. La que sigue es su formulación más genérica:

PRINT#15, "COPY:<nombre nuevo>=<nombre antiguo 1>[, <nombre antiguo 2>, <nombre antiguo 3>, <nombre antiguo 4>]"

Si sólo se desea cambiar el nombre de un archivo, sin modificar la información que contiene, hay que recurrir al comando RENAME, tal como se indica en las siguientes líneas:

PRINT#15, "RENAME:<nombre nuevo>=<nombre antiguo>"

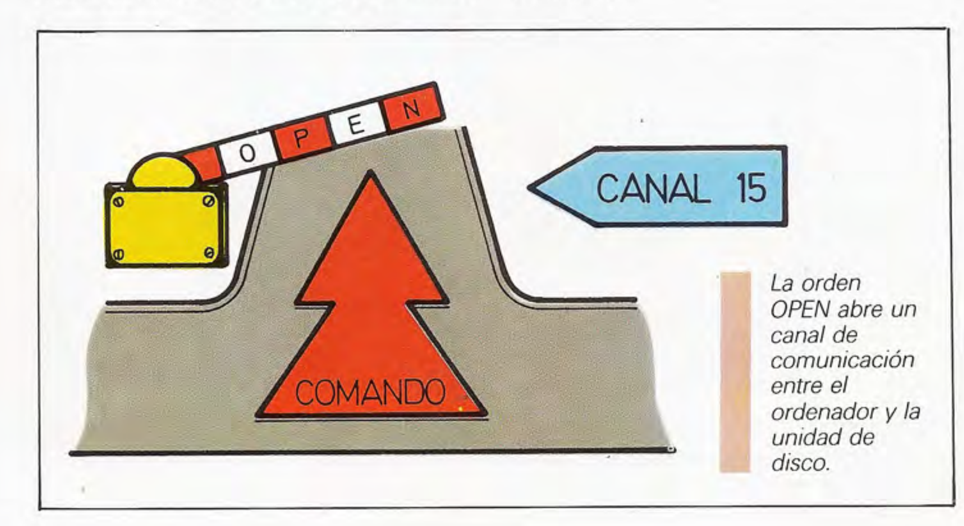
Un ejemplo práctico es el siguiente:

PRINT#15, "RENAME:JOSE=PEPE"

Para borrar un archivo del disco se dispone del comando SCRATCH, el cual hay que complementarlo con el nombre o nombres de los archivos a borrar.

Desde luego, en la formulación de SCRATCH está permitido el uso de las referencias ambiguas «*» y «?». Su formato general será, por tanto:

PRINT#15, "SCRATCH:<nombre1$> [, <nombre 2$>, ...]"




*La orden OPEN abre un canal de comunicación entre el ordenador y la unidad de disco.*

Si al ejecutar algún comando se produce en el disco alguna condición de error, no será posible realizar en él una nueva acción hasta que no sea inicializada de nuevo la unidad de disco, dejándola en el mismo estado en que se encontraba al conectarla. El comando que realiza esta misión es el siguiente:

PRINT#15, "INITIALIZE"

Finalmente, hay que presentar un último comando de disco, VALIDATE, cuya función es la de reorganizar los archivos residentes en el disco flexible. La desorganización tiene su origen en las sucesivas grabaciones y borrados de archivos que darán lugar a la existencia de bloques libres entre archivos. Con la ejecución de VALIDATE también se liberan los bloques utilizados por archivos que no han sido correctamente cerrados y que, por lo tanto, no son accesibles. Este nuevo comando se introduce de la forma siguiente:

PRINT#154, "VALIDATE"

Todos los comandos de discos pueden formularse de forma abreviada mediante su inicial; así, el último comando descrito podría introducirse como sigue:

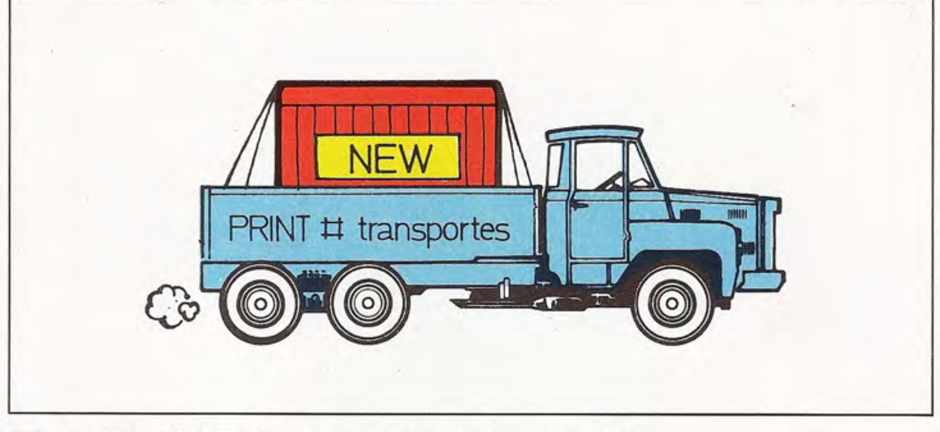
PRINT#15, "V"

## Archivos secuenciales

Los archivos secuenciales se construyen en el disquete transfiriendo a éste toda la información, byte a byte, a través de un buffer. Por esta razón, deben ser escritos y leídos de principio a fin, para lo que previamente se debe establecer un canal de comunicación de datos mediante la orden OPEN:

OPEN <archivo>, <dispositivo>, <canal>, <texto$>

Los parámetros que acompañan a este comando son ya conocidos. Sólo varía el número del canal, que en este caso estará comprendido entre 2 y 14, y el <texto$>. Para la creación de archivos de tipo secuencial, este último parámetro debe tener el siguiente formato:




*La orden PRINT# sirve de transporte para los comandos del disco a través del canal número 15.*

"0" :<nombre>, <tipo>, <dirección>"

La indicación "0 :", debe preceder siempre al nombre del fichero, para así poder acceder a más de dos de los buffers disponibles en la unidad de disco. El <nombre> será una cadena de caracteres válida y constituirá el nombre del archivo a crear o al que se quiere acceder; en él queda autorizado el empleo de referencias ambiguas sólo en las operaciones de lectura de datos. El <tipo> puede ser uno de los siguientes:

PRG — Archivo de programa.
SEQ — Archivo secuencial.
USR — Archivo de usuario.
REL — Archivo relativo.

Todos estos tipos de archivos se crean de la misma forma, diferenciándose únicamente en los comandos que facilitan el acceso a los mismos (recuérdese que incluso el directorio del disco era tratado como si de un archivo de programa se tratara).

La <dirección> debe ser una de las siguientes: "READ" (abreviadamente "R") o "WRITE" (abreviadamente "W"), se-

### OPEN

Abre un canal de comunicación entre el ordenador y el archivo contenido en un determinado dispositivo. También establece un canal de comandos.

*Formato:* OPEN <archivo>, <dispositivo>, <canal>, <texto$>

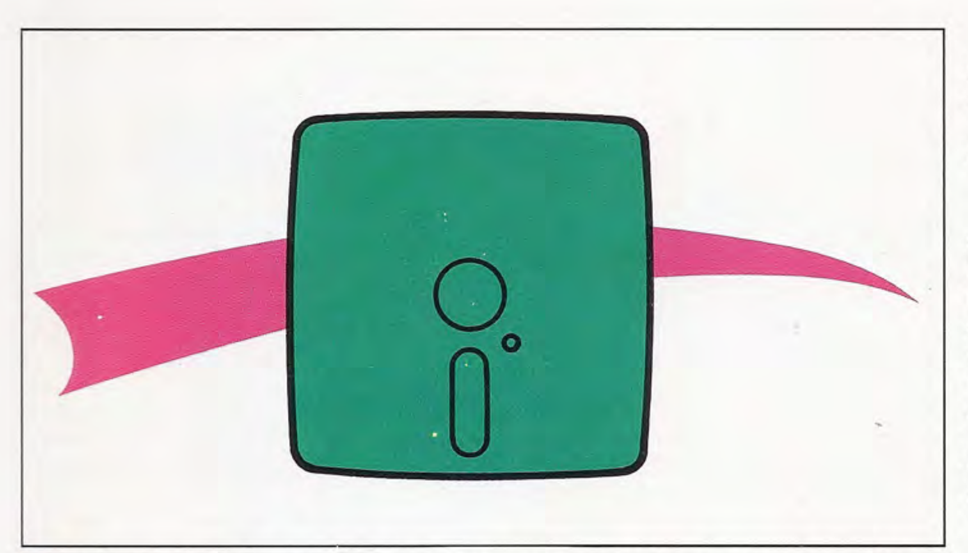
*Ejemplos:* OPEN 15, 8, 15A$
OPEN N1, N2, N3

### PRINT #

Envía la lista de variables que figuran en su argumento a través del canal abierto para el archivo especificado. También puede enviar comandos.

*Formato:* PRINT# <archivo>, <lista de variables>

*Ejemplos:* PRINT# 15, "NEW : DISCO1"
PRINT# 5, A$; B$; C$; D$



*PRINT# e INPUT# actúan para los archivos en disco de forma análoga a como lo hacen PRINT e INPUT para la pantalla y teclado respectivamente.*

gún se quiera acceder al archivo para escribir o leer información en el mismo.

Una vez abierto el canal para la comunicación, hay que utilizar los comandos PRINT# <archivo> o INPUT# <archivo> según se quiera introducir datos en el fichero o leerlos del mismo para depositarlos en la memoria central del ordenador.

El funcionamiento de ambos comandos coincide con el tradicional de las órdenes PRINT e INPUT, hasta el punto de que se respetan los signos de puntuación que separan a las distintas variables que los acompañan. Así, un punto y coma («;») situará los datos uno a continuación del otro, mientras que si se separan por comas («,») se dejarán los espacios en blanco necesarios, entre variable y variable en el disquete, de igual forma que si se fuera a representar en la pantalla.

Como ejemplo ilustrativo del acceso a un archivo secuencial, el programa adjunto, denominado "ERROR DE DISCO", utiliza el comando INPUT# para obtener la información del canal de errores de la unidad de disco.

**LOAD**

Carga el programa nombre$ desde el dispositivo especificado.

*Formato:* LOAD <nombre$>, <dispositivo>, <comando>

*Ejemplos:* LOAD "PRUEBA", 8
LOAD A$, 8, 1

**SAVE**

Almacena un programa en el dispositivo especificado y le asigna el nombre: nombre$.

*Formato:* SAVE <nombre$>, <dispositivo>, <comando>

*Ejemplos:* SAVE "TEST", 8, 1
SAVE B$, D, C

Otro comando utilizado para la lectura de datos de un archivo secuencial es el que adopta el formato:

GET# <archivo>

Su única diferencia con INPUT# consiste en que GET# lee los datos byte a byte (de carácter en carácter). Esta característica lo hace más útil cuando no se conoce con exactitud el contenido ni el formato de un archivo. Un ejemplo ilustrativo lo aporta el programa "LEER ARCHIVO" cuyo listado se adjunta y cuya funcionalidad es la de leer el contenido de cualquier archivo residente en disco.

## Los archivos aleatorios

La unidad de disco de Commodore admite la operación con dos tipos de archivos aleatorios: los así llamados en la nomenclatura Commodore, cuya principal aplicación se encuentra en el lenguaje máquina, y los archivos relativos, más adecuados para manejar información de cualquier tipo. Como ya se ha mencionado, el disquete es dividido por el DOS en pistas y sectores, disponiendo cada sector de un total de 256 bytes libres para el almacenamiento de datos.

En el caso de los archivos aleatorios es posible acceder directamente a cada uno de los bloques del disco, mediante comandos similares a los ya descritos. Estos se envían a través del canal 15 de comandos, apoyándose en las órdenes "OPEN" y "PRINT#". Por lo demás, es necesario ahora abrir un segundo canal para el trasvase de datos, lo que se logra con la siguiente orden:

OPEN <archivo>, <dispositivo>, <canal>, "#"

Todos los parámetros son ya conocidos, excepto el referenciado por el símbolo "#" que indica el buffer de datos a utilizar para el archivo aleatorio. Este símbolo puede ir seguido por un número, por ejemplo el #2, lo que señala que se quiere utilizar el buffer número 2.

El comando que permite el trasvase de un determinado bloque del disco al canal de datos abierto es BLOCK-READ, el cual adopta el siguiente formato:

PRINT# 15, "BLOCK-READ:" <canal>; <dispositivo>; <pista>; <bloque>

Junto a los parámetros ya conocidos, hay que especificar también la pista y el bloque exacto que se desea leer. Hecho esto, la información se puede obtener ya recurriendo a las órdenes "INPUT#" o "GET#", lo que es más habitual.

El comando opuesto al descrito es BLOCK-WRITE, cuyo formato es idéntico al anterior. Su ejecución debe ser posterior al envío de los datos por el canal mediante la orden "PRINT#".

Otros dos comandos opuestos son BLOCK-ALLOCATE y BLOCK-FREE. El primero se encarga de reflejar en el BAM que el bloque indicado está ocupado por datos y no está disponible a partir de ese momento. El segundo realiza la operación contraria, es decir, libera un bloque cuyo contenido ya no se desea conservar. Por ejemplo:

PRINT# 15, "B—A : "0; PISTA; BLOQUE

Si se intenta utilizar un bloque ya ocupado, se generará un mensaje de error que debe ser leído del canal de errores. Junto al mensaje de error, 65: NO-BLOCK (bloque ocupado), se indicará al usuario el siguiente bloque libre del disco.

El inconveniente que presenta este tipo de archivos es que resulta muy difícil seguir la pista de los bloques que se van utilizando. Para evitar dicho problema se suele crear paralelamente un archivo de tipo secuencial cuyo contenido coincide con la lista de las pistas y bloques usados.

El buffer que se utiliza para el trasvase de datos posee un puntero que seña-

**COMANDOS DEL DISCO**

| Comando | Formulación en el argumento de OPEN o PRINT# | Función |
|---|---|---|
| NEW | "NEW:<nombre>, <identificador>" | Formatea e inicializa un disco. |
| COPY | "COPY:<nom. nuevo>=<nom. antiguo>" | Copia un archivo en disco. |
| RENAME | "RENAME:<nom. nuevo>= <nom. antiguo>" | Cambia de nombre a un archivo. |
| SCRATCH | "SCRATCH:<nombre>" | Borra un archivo del disco. |
| INITIALIZE | "INITIALIZE" | Inicializa la unidad de disco. |
| VALIDATE | "VALIDATE" | Reorganiza los archivos en disco. |
| OPEN | "0:<nombre>, <tipo>, <dirección>" | Abre un fichero secuencial. |
| OPEN | "#" o "#<número de buffer>" | Abre un canal de datos para acceso aleatorio. |
| BLOCK-READ | "BLOCK-READ:" <canal>; <disp.>; <pista>; <bloque> | Traslada un bloque de datos del disco al canal. |
| BLOCK-WRITE | "BLOCK-WRITE:"<canal>; <disp.>; <pista>; <bloque> | Opuesto al anterior. |
| BLOCK-ALLOCATE | "BLOCK-ALLOCATE:"<disp.>; <pista>; <bloque> | Señala en el BAM el bloque utilizado. |
| BLOCK-FREE | "BLOCK-FREE:" <disp.>; <pista>; <bloque> | Libera en el BAM el bloque desechado. |
| BUFFER-POINTER | "BUFFER-POINTER:"<canal>; <posición> | Posiciona el puntero del buffer. |
| OPEN | "<nombre>,L,"+CHR$(longitud registro) | Crea un archivo relativo. |
| PRINT# | "P"CHR$(<canal>+96)CHR$(LSB) CHR$(MSB)CHR$(<posición>) | Posiciona el puntero de un archivo relativo. |




*El puntero del BUFFER de un archivo aleatorio se puede situar en cualquier punto de éste mediante el comando «BUFFER-POINTER».*

```
10 REM ERROR DE DISCO
20 OPEN15, 8, 15
30 INPUT*15, NUM, MENSA$, PISTA, SECTOR
40 PRINT CHR$(147)
45 IF NUM=0 THEN 60
50 PRINT "Error-"; NUM;
60 PRINT"    "; MENSA$
70 PRINT "En pista: "; PISTA; " sector: "; SECTOR
80 CLOSE15 : END
```

```
10 REM LEER ARCHIVO
20 INPUT "Archivo "; A$
30 INPUT "Tipo "; T$
40 OPEN15, 8, 15 : OPEN 2, 8, 2, "0 : "+A$+", "+T$+", R"
50 INPUT*15, NUM, MENSA$, PISTA, SECTOR
60 IF NUM>0 THEN PRINT NUM, MENSA$, PISTA, SECTOR : END
70 GET*2, N$
80 IF ST=0 THEN 100
90 CLOSE15, 2 : END
100 PRINTASC(N$);
110 GOTO 50
```

la al último byte de datos escrito o al siguiente que se va a leer. Es posible hacer que este puntero señale a la deseada porción de datos de un bloque mediante el comando BUFFER-POINTER, cuyo formato es el siguiente:

PRINT# 15, "BUFFER-POINTER: "<canal>, <posición>

Su ejecución permite dividir cada bloque en registros independientes y éstos, a su vez, en campos que proporcionan estructuras de almacenamiento más efectivas. Y todo ello sin más que asignar una determinada <posición> dentro del bloque a cada uno de estos campos y registros.

## Los archivos relativos

Este nuevo tipo de archivos se caracteriza por permitir la estructuración de los datos en registros, y éstos a su vez en campos, lo que potencia su utilidad a la hora de manejar datos en régimen de acceso directo.

Para crear un archivo relativo se utiliza el siguiente formato:

OPEN <archivo>, <dispositivo>, <canal>, "<nombre$>, L, "+CHR$ (<longitud del registro>)

El DOS establecerá una serie de punteros que señalarán a los sectores adyacentes a cada registro. De esta forma, cada sector puede apuntar hasta 720 registros, que pueden incluir hasta 254 caracteres por registro, lo que convertiría al disquete completo en un único archivo relativo.

Una vez creado un archivo relativo, su apertura para el acceso resulta ya más simple, puesto que no hay que especificar más que su nombre, junto a los restantes parámetros típicos del comando OPEN. Para acceder a cualquier registro del archivo abierto hay que empezar posicionando el puntero del archivo en el registro y posición deseada, lo que se hace con la siguiente orden:

PRINT# <archivo>, "P" CHR$ (<canal1>+96) CHR$(<registro LSB>) CHR$(<registro MSB>) CHR$(<posición>)

En ella, <registro LSB> corresponde al byte menos significativo del número de registro al que se quiere acceder y <registro MSB> al byte más significativo. La necesidad de dos bytes para este dato deriva de que con un byte sólo se pueden representar los números del 0 al 255, y no hay que olvidar que un archivo en la unidad de disco 1541 de Commodore puede contener más de 700 registros.

Para calcular el valor de estos dos bytes puede procederse como sigue. Por ejemplo, si el registro al que se quiere acceder es el número 520, el MSB será MSB=INT (520/256) y el LSB será entonces: LSB=520—MSB*256. Al evaluar ambas expresiones se obtienen los siguientes valores: MSB=2 y LSB=8.

El parámetro <posición> se refiere, naturalmente, a la posición dentro de cada registro. Su colaboración permite apuntar a un campo determinado dentro del registro.

El acceso al fichero relativo se hará, por lo tanto, en función del número del registro deseado. Si se quiere acceder por medio de una clave o código a cada registro, habrá que construir paralelamente un archivo secuencial que relacione cada clave con el número de registro correspondiente.

# Archivos en el Spectrum

## Creación y tratamiento de archivos en Microdrive



Los Microdrives son unidades de almacenamiento masivo, en forma de cartuchos de cinta magnética, enrollada de tal forma que no tiene fin. Estos dispositivos presentan ciertas ventajas respecto a los tradicionales magnetófonos a casete, aunque sus características distan mucho de las que brindan las unidades de disco.

Las unidades de Microdrive se conectan al Spectrum a través del Interface-1. Este dispositivo, además de controlar a los Microdrives, potencia al Spectrum con la aportación de una red local y un interface RS/232.

La primera operación a realizar, tanto con un disco como con un cartucho, es el formateado del soporte magnético. Esta es una operación durante la cual se crea además el directorio. Para ello es necesario ejecutar el comando FORMAT, cuya formulación general es:

FORMAT*"m";n;<nombre>

en donde:

n: es el número de unidad de Microdrive en el que se encuentra el cartucho que se desea formatear (puede haber hasta ocho unidades conectadas).

<nombre>: nombre que se desea otorgar al cartucho. Puede ser una cadena de caracteres encerrada entre comillas, o una variable de cadena.

Otra operación fundamental para el manejo de los cartuchos es la obtención de su catálogo o directorio. Esta operación se realiza mediante el comando CAT, cuya formulación es:

CAT n

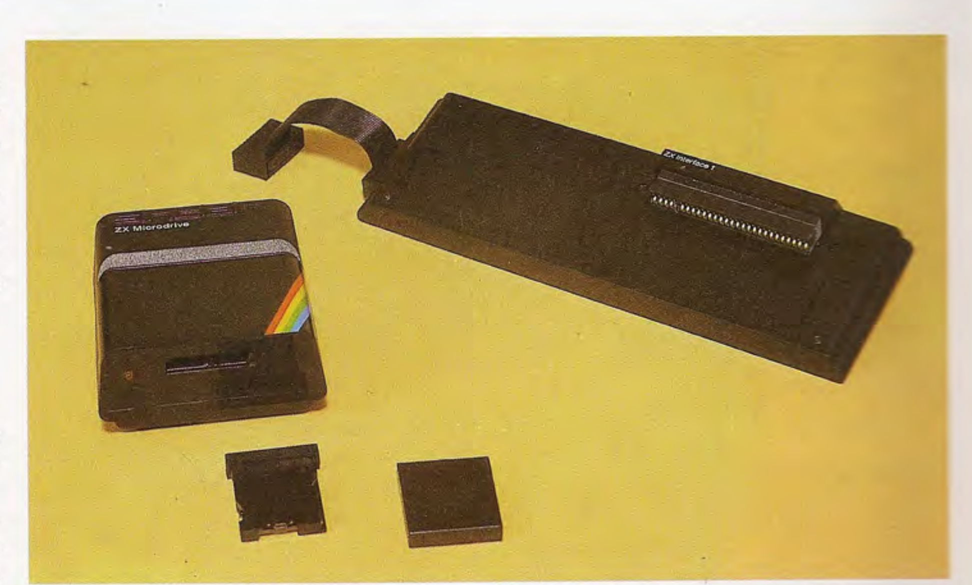
donde n es el número de Microdrive en el que reside el cartucho del que se desea obtener el catálogo.

Este comando puede utilizarse también para comprobar si una determinada operación se ha realizado correctamente.

Los nombres de los archivos creados en Microdrive siguen aproximadamente las mismas reglas aplicadas a los archivos en casete. Esto es: el nombre ha de constar de no más de 10 caracteres. No obstante el nombre puede incluir cualquiera de los «tokens» propios del Spectrum, con lo que, aparentemente, el nú-




*Aun cuando la opción básica para el almacenamiento externo de información en la familia Sinclair la constituyen los magnetófonos a casete, la mayor velocidad de acceso del Microdrive, sumado a su precio moderado, han convertido a este periférico en una frecuente alternativa.*




*Las unidades de Microdrive constituyen un original periférico de almacenamiento, cuyo soporte son cintas magnéticas alojadas en un diminuto cartucho de material plástico.*

mero total de caracteres se ve incrementado.

## Comandos generales para la manipulación de archivos

Los comandos dedicados a la manipulación de archivos deben permitir las siguientes operaciones básicas: grabar, leer, modificar, verificar, mezclar y borrar archivos.

En el caso del Microdrive, los comandos al efecto admiten las mismas extensiones que sus homólogos destinados al manejo del casete; esto es: CODE para bytes, SCREEN$ para el caso particular de la pantalla y LINE para la autoejecución de un programa BASIC.

Para grabar un programa en un cartucho alojado en la unidad de Microdrive se utiliza el comando SAVE, aunque con una formulación distinta de la habitual:

SAVE*"m";n;<nombre>(<extensiones>)

en donde:

n señala el número de unidad de Microdrive en la que se va a realizar la grabación;

<nombre> corresponde al nombre que se desea dar al archivo sujeto a la operación ordenada.

<extensiones>: pueden ser, opcionalmente, LINE, CODE o SCREEN$.

Para leer un archivo de programas ha de utilizarse el comando LOAD, cuya formulación mantiene un gran paralelismo con la propia de SAVE:

LOAD *"m";n;<nombre>(<extensiones>)

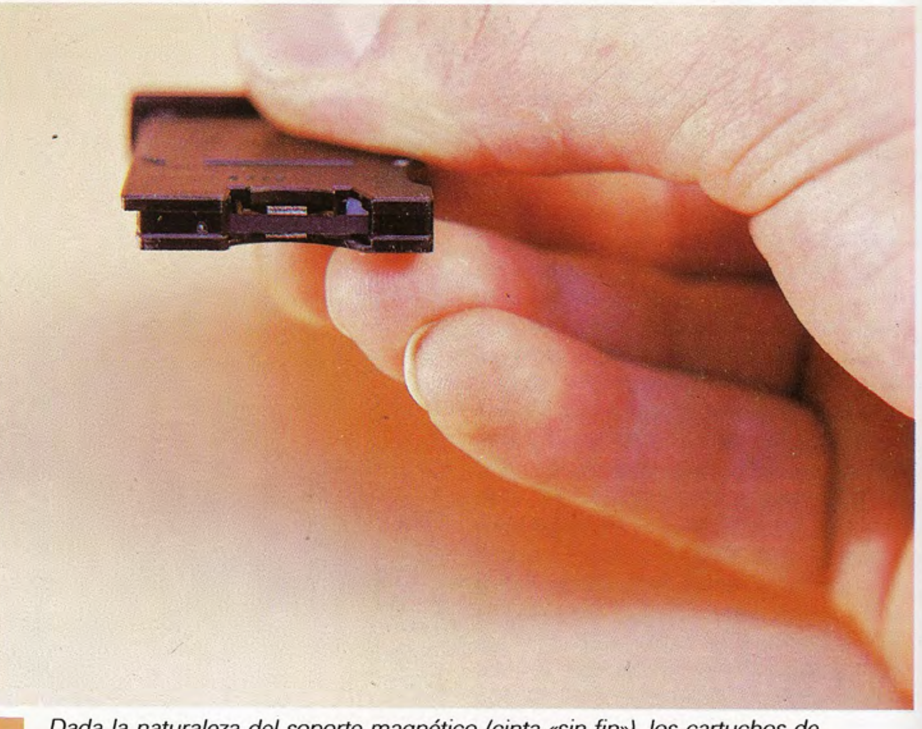
en donde:

n indica la unidad de Microdrive en uso.

<nombre> es el nombre del archivo de programa que se desea leer.

<extensiones>: pueden ser, opcionalmente, CODE ó SCREEN$.

Una vez efectuada la grabación de un programa, es posible que el usuario quiera asegurarse de que ésta se ha efectuado correctamente. Para ello cuenta con el comando VERIFY. Este comprobará que el archivo se ha grabado correctamente comparando la información en él contenida con la que se encuentra presente en memoria. Desde luego, su ejecución sólo será eficaz si permanece inalterado el contenido de la memoria interna que se desea cotejar.



*Dada la naturaleza del soporte magnético (cinta «sin fin»), los cartuchos de Microdrive son soportes de tipo exclusivamente secuencial, semejantes a las tradicionales casetes.*

**FORMAT**

Permite formatear un cartucho de Microdrive.

*Formato:* FORMAT* "m"; n; < nombre>

*Ejemplo:* FORMAT* "m"; 1; "NOMBRE"

**CAT**

Lee el directorio de un cartucho.

*Formato:* CAT n

*Ejemplo:* CAT 1

## SAVE*

Almacena un programa en Microdrive.

*Formato:* SAVE* "m"; n; <nombre>[<extensiones>]

*Ejemplos:* SAVE* "m"; 1; "PROG-1"
SAVE* "m"; 2; "ROG" LINE 10

## LOAD*

Lee un programa residente en cartucho.

*Formato:* LOAD* "m"; n; <nombre>[<extensiones>]

*Ejemplo:* LOAD* "m"; 1; "ROL"

Este hecho justifica el escaso empleo del referido comando con la extensión SCREEN$. Su formulación general es la siguiente:

VERIFY *"m";n;<nombre>(<extensiones>)

siendo:

n el número de la unidad de Microdrive donde se encuentra el cartucho activo,

<nombre> el nombre del programa grabado en Microdrive con el que se desea comparar el contenido actual de la memoria.

<extensiones>: pueden ser CODE o SCREE$, opcionalmente.

En ciertos casos es necesario utilizar dentro del programa en curso una rutina que se confeccionará aparte o bien que se encuentra inmersa en otro programa. En tal caso, hay que recurrir al comando MERGE, el cual permite leer un programa y «mezclarlo» con el que se encuentra en memoria. Dicha «mezcla» se realiza sin problemas siempre que los números de línea de ambos programas no coincidan.

No obstante, si los números coinciden se producirá una situación anómala, derivada del hecho de que no pueden coexistir en memoria dos líneas con el mismo número. En tal caso, de efectuarse la operación MERGE, desaparecerán las líneas coincidentes del programa que se encontraba en memoria.

La formulación del comando MERGE es la que sigue:

MERGE *"m";n;<nombre>

en donde:

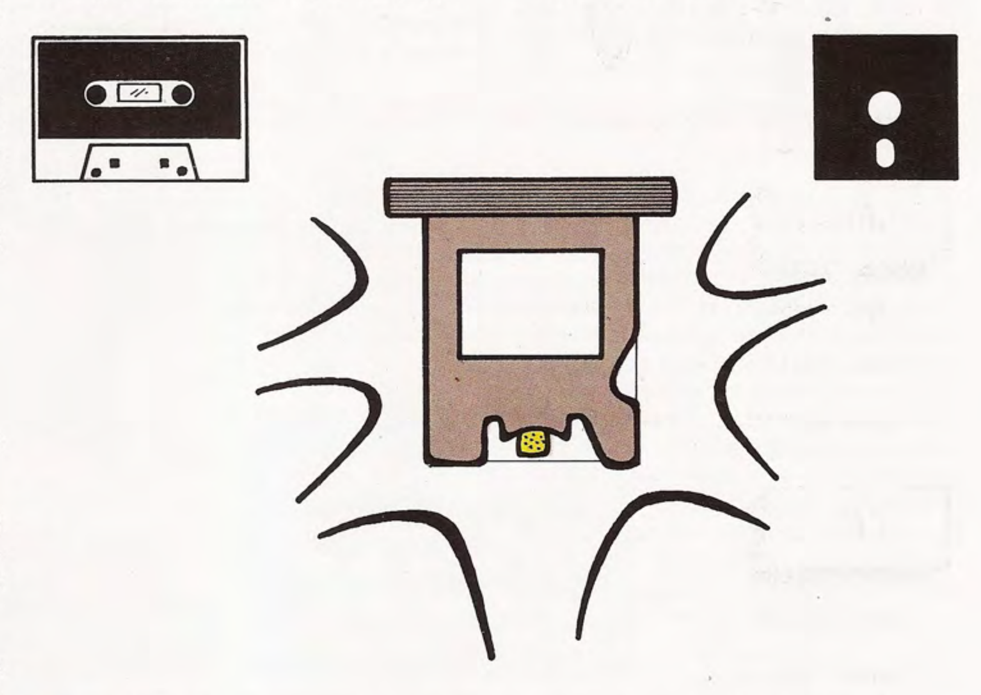
n es el número del drive que se desea emplear;

<nombre> corresponde al nombre del archivo cuyo contenido se desea mezclar con el programa que se encuentra en memoria.

Por último, hay que hacer mención de un comando cuyo empleo será desconocido para los que sólo hayan empleado unidades de casete. Se trata del comando ERASE, cuya misión es borrar un archivo. Su utilidad es manifiesta en los Microdrives ya que, a diferencia de lo que ocurre con los casetes, en los cartuchos no es posibles situar la cabeza lecto-grabadora al principio del programa no deseado y grabar sobre el mismo la nueva información.

La sintaxis de una instrucción de esta índole es:

ERASE "m";n;<nombre>



*Los Microdrives constituyen una alternativa adicional, entre el disco y el casete, para el almacenamiento masivo de información en el Spectrum.*

en donde:

n es el número de la unidad donde se encuentra el cartucho;

<nombre> es el nombre del archivo que se desea eliminar.

**OPEN#**

Abre un archivo de datos.

*Formato:* OPEN# C; "m"; n; <nombre>

*Ejemplo:* OPEN# 4; "m"; 1; "DATOS-1"

**CLOSE#**

Cierra un archivo de datos.

*Formato:* CLOSE# n

*Ejemplo:* CLOSE# 5

## Comandos para la manipulación de archivos de datos

Debido a la propia naturaleza de los Microdrives y a la estructura del soporte utilizado (cinta magnética continua), esta unidad de almacenamiento sólo permite trabajar con archivos de tipo secuencial. Esta es su principal limitación.

En los capítulos precedentes dedicados al estudio de los archivos secuenciales, se apuntaron las cuatro operaciones básicas que pueden realizarse en un archivo de este tipo: apertura, cierre, escritura y lectura.

La apertura del archivo corre a cargo del comando OPEN, cuya formulación adopta en este caso el siguiente aspecto:

OPEN #c;"m";n;<nombre>

en donde:

c corresponde al número de canal seleccionado (puede haber varios canales abiertos siempre y cuando no coincidan los números).

<nombre> es el nombre del archivo que se desea abrir;

n es el número de la unidad de Microdrive utilizada.

Tras efectuar cualquier operación con un archivo es imprescindible cerrarlo; de lo contrario pueden ocurrir pérdidas irremediables de información. El comando preciso es CLOSE:

CLOSE #n

donde n es el número del canal asociado al archivo que se desea cerrar.

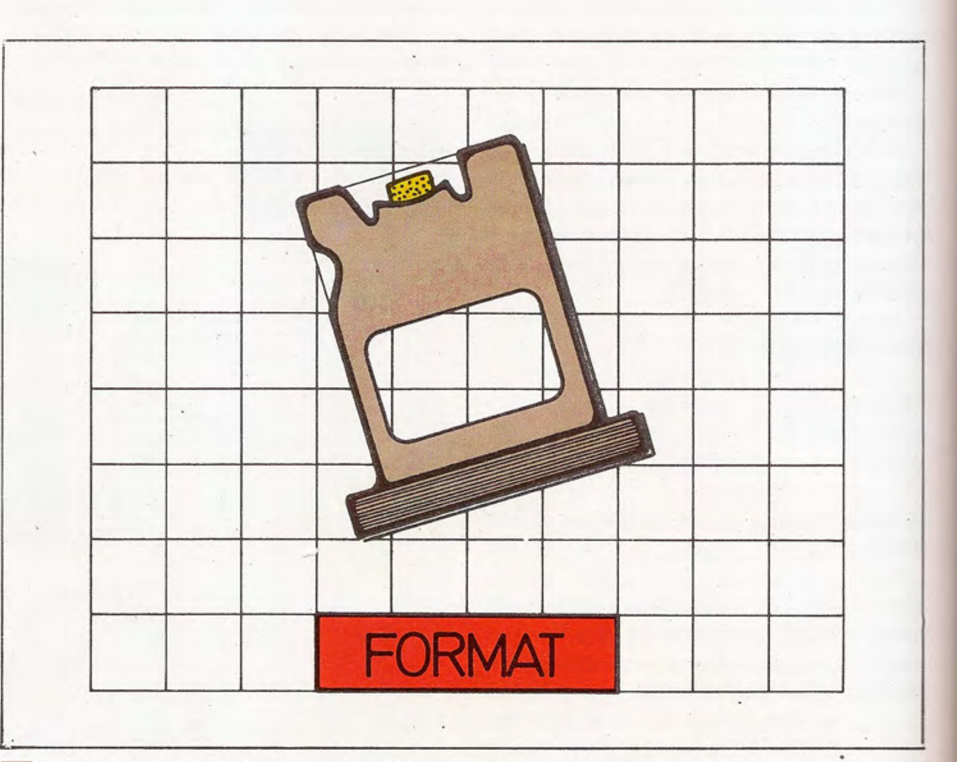
Para introducir datos en el archivo en uso se utiliza el recurrido comando PRINT, aunque con una sintaxis distinta de la habitual:

PRINT #n;<datos>

en donde:

n es el número de canal asociado al archivo con el que se está trabajando, y

<datos> es la lista de datos que se de-




*Al igual que ocurre con los discos, los cartuchos de Microdrive deben ser previamente formateados para que sea posible realizar en ellos cualquier operación de lectura o escritura.*

sea grabar en un registro del archivo secuencial.

Por último, para leer los datos grabados en el archivo se emplea el comando INPUT:

INPUT #n;<lista de variables>

en donde:

n corresponde al número de canal asociado con el archivo, y

<lista de variables> a la lista de variables a las que se otorgarán los datos que se leen del archivo.

Existe otra función adicional orientada a la lectura de un archivo. Se trata de INKEY$#, cuyo formato es el que sigue:

INKEY$ #n

Este comando permite la lectura de los caracteres grabados en un archivo, de uno en uno.

**PRINT#**

Escribe en un archivo de datos.

*Formato:* PRINT# n <datos>

*Ejemplos:* PRINT# 3; A$, "**"

**INPUT#**

Lee de un archivo de datos.

*Formato:* INPUT# n <datos>

*Ejemplo:* INPUT# n; A$

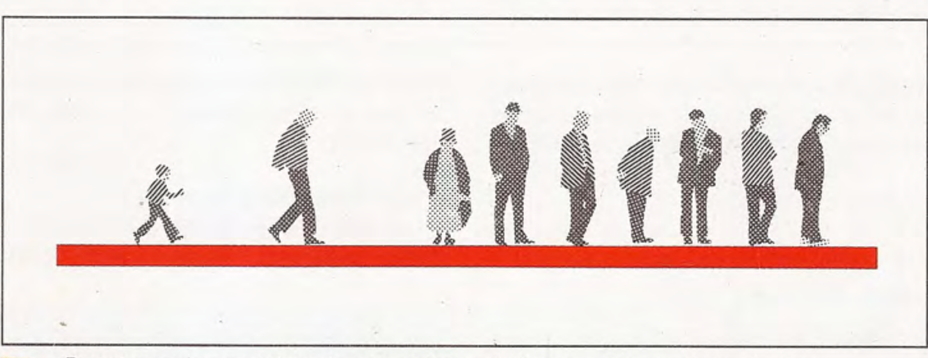
## Ejemplos prácticos

*Creación de un archivo secuencial*

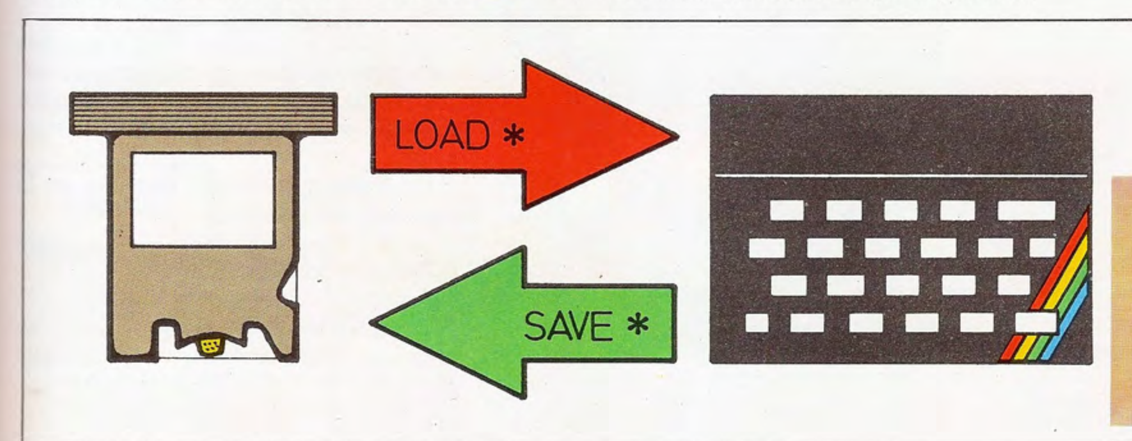
El primer ejemplo es un programa muy sencillo que permite la creación de un archivo secuencial. Los datos se solicitan a través de instrucciones INPUT y el programa se encarga de irlos introduciendo en el archivo externo abierto al efecto.

El final del archivo se marca por medio de un elemento cuyo contenido es fijo y que consiste en tres asteriscos. Si no se marcara el final del archivo y se intentara leer su contenido con posterioridad, se produciría un error que detendría la ejecución del programa.

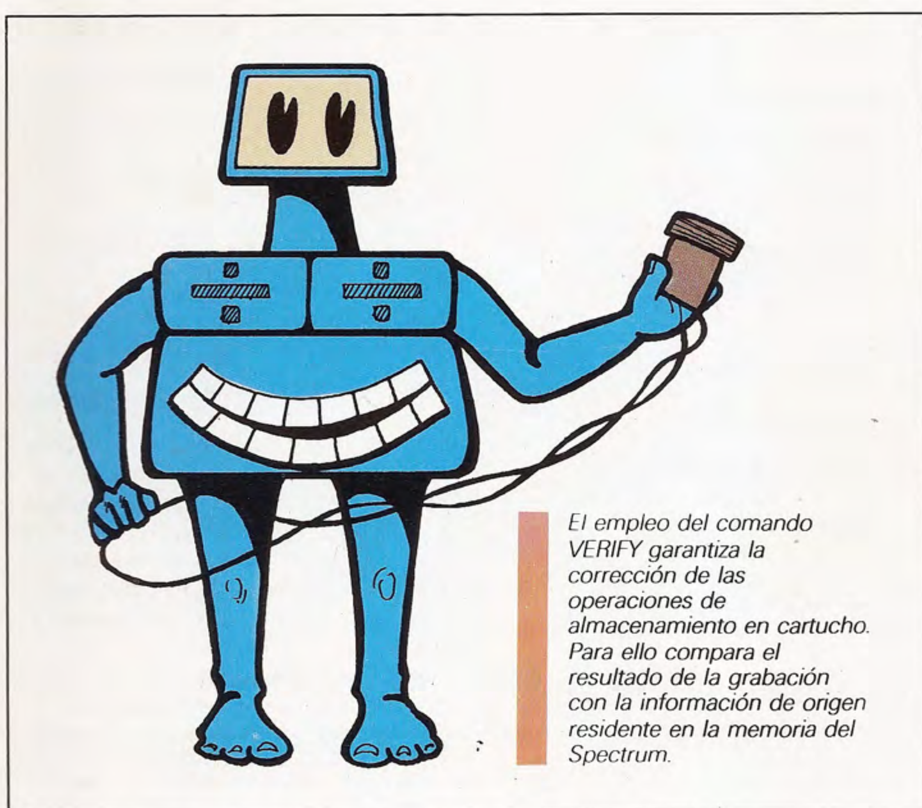
Cualquier operación con un archivo



*Por su propia naturaleza, los cartuchos admiten tan sólo archivos de tipo secuencial. En ellos los datos se almacenan uno tras otro, en estricto orden, por lo que para acceder a cualquiera de ellos es preciso pasar por todos los que lo preceden.*




*Los tradicionales comandos SAVE y LOAD (seguidos por un asterisco y los correspondientes parámetros) permiten el transvase de información entre el Spectrum y el Microdrive.*




*El empleo del comando VERIFY garantiza la corrección de las operaciones de almacenamiento en cartucho. Para ello compara el resultado de la grabación con la información de origen residente en la memoria del Spectrum.*

empieza por su apertura. Tal acción lleva implícita además la propia creación del archivo puesto que éste no existe:

```
10 OPEN #5; "m";1;"DATOS"
```

A continuación, hay que establecer el medio de captación de los datos a grabar; por ejemplo, mediante una instrucción INPUT:

```
20 INPUT "INTRODUZCA EL DATO"; LINE A$
```

Para indicar al programa que no se desea introducir más datos en el archivo, es suficiente con responder al INPUT con la palabra FIN. Al detectar esta palabra, el programa pasará a ejecutar la rutina de cierre del archivo con el que se está trabajando:

```
30 IF A$="FIN" THEN GOTO 60
```

La instrucción de la línea 40 es la encargada de grabar en el archivo el dato introducido. Obsérvese que el número de canal asociado ha de coincidir con el especificado en el momento de abrir el archivo en cuestión:

```
40 PRINT #5; A$
```

Para proseguir con la introducción de datos es necesario que la secuencia del programa regrese a la instrucción de la línea 20, lo cual se logra por medio de un simple GOTO:

```
50 GOTO 20
```

Finalmente, hay que añadir las instrucciones adecuadas para cerrar el archivo. En primera instancia habrá que escribir en él los caracteres *** que servirán para detectar el final del archivo. Tras ello puede ya ejecutarse la oportuna orden CLOSE:

```
60 PRINT#5;"***"
70 CLOSE#5
80 STOP
```

***Rutina de lectura***

El siguiente ejemplo es una rutina adecuada para leer los datos que fueron introducidos en el archivo por medio del anterior programa.

La rutina en cuestión está preparada para detectar el fin del fichero (caracteres ***) y evitar de esta forma la interrupción del programa por efecto de un error.

La primera operación consiste en la apertura del archivo:

```
10 OPEN #5;"m";1;"DATOS"
```

Tras ello, se leerá un elemento del archivo por medio de la siguiente instrucción:

```
20 INPUT #5;A$
```

## INKEY$#

Lee un carácter de un archivo.

*Formato:* INKEY$# n

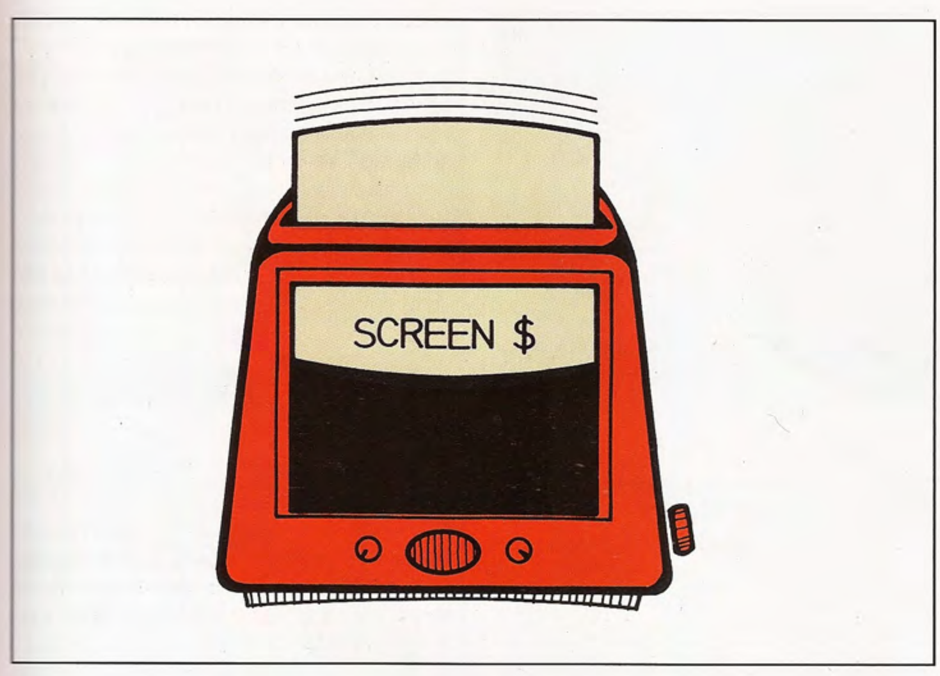
*Ejemplo:* INKEY$# 4

## VERIFY*

Permite verificar si una grabación se ha efectuado correctamente.

*Formato:* VERIFY* "m"; n; <nombre>[<extensiones>]

*Ejemplo:* VERIFY* "m"; 1; "DATO"




La opción SCREEN$ aplicada a los comandos LOAD*, SAVE* o VERIFY* permite manipular archivos cuyo contenido es información de pantalla.

En la línea 30 se detectará si el dato leído coincide con el indicador de fin de archivo:

```
30 IF A$="***" THEN GOTO 60
```

Para mostrar por pantalla el elemento extraído por efecto de la operación de lectura, basta con imprimir la variable A$ por medio de la instrucción siguiente:

```
40 PRINT A$
```

En este punto, es preciso que la secuencia del programa regrese a la línea 20 para que sea leído un nuevo dato del archivo:

```
50 GOTO 20
```

Y, por último, hay que proceder al cierre del archivo y, cómo no, detener la ejecución del programa:

```
60 CLOSE #5
70 STOP
```

### *Rutina de actualización*

La tarea más compleja que se puede realizar con un archivo secuencial es la actualización de su contenido. Evidentemente, ésta no puede efectuarse de una forma directa, de ahí que haya de recurrirse a otros artificios. La solución reside en actualizar el archivo actual copiando los elementos ya existentes en un nuevo archivo, y grabando a continuación los nuevos elementos.

Como de costumbre, se empezará abriendo los archivos implicados en la

## MERGE*

Mezcla dos programas.

*Formato:* MERGE* "m"; n; <nombre>

*Ejemplo:* MERGE* "m"; 1; "ALIEN"

## ERASE

Borra un archivo del disco.

*Formato:* ERASE "m"; n; <nombre>

*Ejemplo:* ERASE "m"; 1; "GOLF"

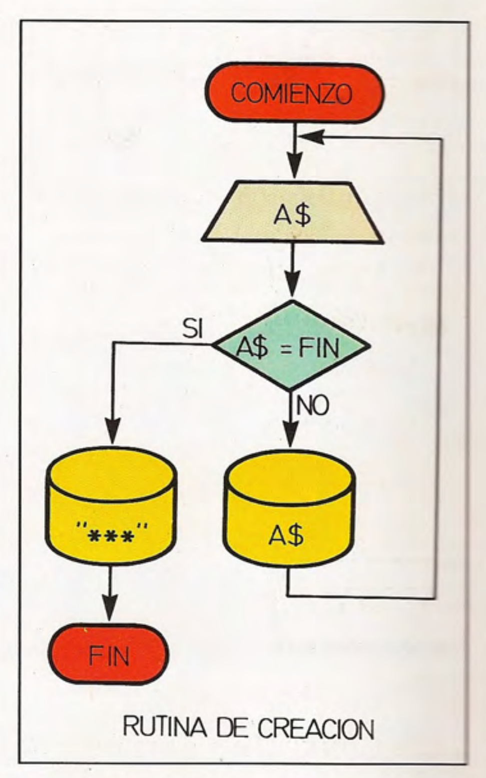



*Diagrama de flujo asociado a la rutina de creación de un archivo secuencial descrita en el texto.*

## Rutina 1: Creación de un archivo secuencial

```
 5 REM RUTINA DE CREACION
10 OPEN#5; "m"; 1; "DATOS"
20 INPUT "INTRODUZCA EL DATO"; LINE A$
30 IF A$="FIN" THEN GOTO 60
40 PRINT#5; A$
50 GOTO 20
60 PRINT#5; "***"
70 CLOSE#5
80 STOP
```

## Rutina 2: Lectura de un archivo secuencial

```
 5 REM RUTINA DE LECTURA
10 OPEN#5; "m"; 1; "DATOS"
20 INPUT#5; A$
30 IF A$="***" THEN GOTO 60
40 PRINT A$
50 GOTO 20
60 CLOSE#5
70 STOP
```

## Rutina 3: Actualización de un archivo secuencial

```
  5 REM RUTINA DE ACTUALIZACION
 10 OPEN#5; "m"; 1; "DATOS"
 20 OPEN#4; "m"; 1; "DATOS.1"
 30 INPUT#5; A$
 40 IF A$="***" THEN GOTO 70
 50 PRINT#4; A$
 60 GOTO 30
 70 INPUT "INTRODUZCA EL DATO"; LINE A$
 80 IF A$="FIN" THEN GOTO 110
 90 PRINT#4; A$
100 GOTO 70
110 PRINT#4; "***"
120 CLOSE#5: CLOSE *4
130 STOP
```

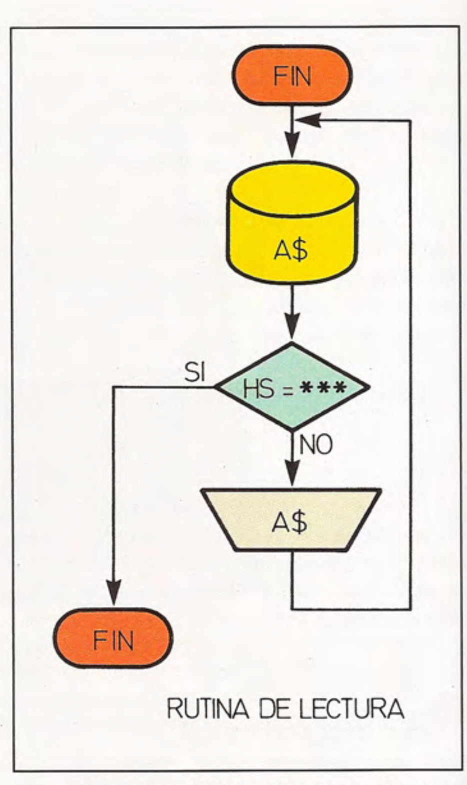





*Diagrama de flujo correspondiente a la rutina de lectura de un archivo secuencial.*

operación; dos en el caso que nos ocupa:

```
10 OPEN #5;"m";1;"DATOS"
20 OPEN #4;"m";1;"DATOS.1"
```

Acto seguido, hay que instrumentar los medios necesarios para leer los datos del archivo de origen, uno a uno, y grabarlos en el archivo de destino. Por supuesto, cuando en el archivo de origen se detecte la marca de fin de archivo (***), habrá que saltar el bloque de instrucciones que gestionarán la introducción en el archivo de destino de los nuevos datos implicados en el proceso de actualización. Las acciones descritas quedan en manos de las siguientes instrucciones:

```
30 INPUT#5;A$
40 IF A$="***" THEN GOTO 70
50 PRINT#4;A$
60 GOTO 30
```

La zona encargada de la introducción de los nuevos datos, que ampliarán los ya existentes, y de colocar la marca de fin del archivo en su nuevo emplazamiento coincide con la que sigue:

```
 70 INPUT "INTRODUZCA EL DATO"; LINE A$
 80 IF A$="FIN" THEN GOTO 110
 90 PRINT#4;A$
100 GOTO 70
110 PRINT#4;"***"
```

Por último, sólo queda cerrar los archivos utilizados y detener la ejecución del programa:

```
120 CLOSE#5:CLOSE#4
130 STOP
```

# Ficheros en los AMSTRAD

## La coexistencia de CP/M y AMSDOS




na característica primordial de los ordenadores Amstrad es su capacidad de disponer del sistema operativo CP/M, el cual abre las puertas de un nuevo universo de aplicaciones.




*El Amstrad es un ordenador que se ha hecho tremendamente popular en los últimos tiempos, gracias a su arquitectura integrada: unidad central, monitor y unidad de disco o casete se venden conjuntamente a un precio muy ventajoso.*

## El sistema de discos

Los modelos CPC664 y CPC6128 de la familia Amstrad incorporan en la parte derecha del teclado una unidad de disco de tres pulgadas, mientras que en el CPC464 este espacio es ocupado por un casete convencional. Para poder disfrutar en este último modelo de las prestaciones del sistema de discos, es necesario conectarle una unidad adicional a través del conector correspondiente.

También es posible conectar una segunda unidad de disco a cualquiera de los modelos Amstrad; ésta puede elegirse para trabajar con discos de tres pulgadas o de cinco y un cuarto. Cada cara de los discos, independientemente de su tamaño, tiene capacidad de almacenar unos 170K de información; aunque ello parezca mucho a simple vista, dicha capacidad puede resultar escasa para trabajar con algunos programas en CP/M, o para mantener una base de datos algo extensa.

Nada más conectar el aparato, el sistema de discos queda bajo el mando del AMSDOS (acrónimo para «AMStrad Disk Operating System») el cual ofrece una serie de comandos, algunos de ellos muy parecidos a los del CP/M, para la gestión de ficheros y directorios.

Para pasar el control al CP/M se teclea un comando específico de AMSDOS; a partir de entonces se dispone de las órdenes típicas del CP/M.

## AMSDOS, un S.O. desde BASIC

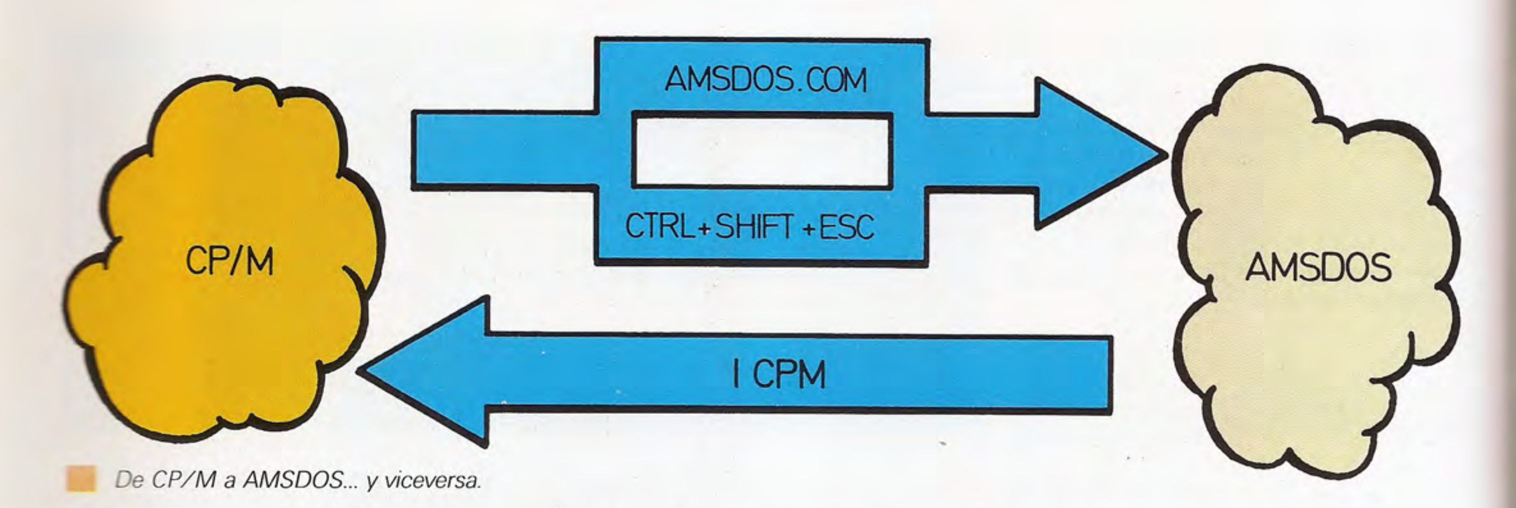
Como se ha comentado, AMSDOS está disponible desde el momento en el que lo está el intérprete BASIC; esto es, desde que se conecta el ordenador. Esto es así debido a que los comandos que proporciona el AMSDOS son, realmente, una serie de RSX implementados en RON.

En el Amstrad, el usuario puede crear a su gusto nuevos comandos que siempre van precedidos de una barra vertical. En la mayoría de las ocasiones, estos comandos residirán en la RAM del equipo, por lo que cada vez que se quiera hace uso de ellos deben ser recogidos previamente de cinta o disco. Otra opción consiste en que una vez programados puedan convertirse en parte indeleble del conjunto de comandos que proporciona el equipo sin son grabados en ROM; precisamente esto es lo que ha ocurrido con los RSX que constituyen el sistema operativo AMSDOS.

Pese a que las ventajas que presenta el sistema de discos respecto al de cintas son cuestionables, hay ocasiones en las que disponer de un almacenamiento masivo sobre casetes puede resultar interesante; por ejemplo, cuando se trata de obtener copias de seguridad (backups) de los programas y datos más importantes. En estos casos, guardar las copias en cinta resulta mucho más barato que hacerlo en disco, y el problema del mayor tiempo de acceso a la cinta queda compensado por la esperanza de que no habrá que utilizarlas muy frecuentemente. Al efecto, AMSDOS proporciona los comandos oportunos para




*El benjamín de la familia, el CPC 464, lleva incorporada en la unidad central la unidad de casete. Sus hermanos mayores se han decantado por el disco, más práctico y rápido.*




*De CP/M a AMSDOS... y viceversa.*

«redirigir» las entradas y salidas, bien sea hacia el disco o hacia la cinta. Estos comandos están reflejados en el cuadro adjunto.

En la correspondiente tabla aparecen los restantes comandos que proporciona AMSDOS para la gestión de los ficheros y del espacio en disco. Se observa la presencia del comando |USER, que permite realizar hasta 16 particiones del disco (|USER,0... |USER,15), lo cual resulta especialmente útil cuando la escasez de discos obliga a tener directorios excesivamente poblados. Mediante este comando es posible dividir el disco en secciones para programas BASIC, ficheros en código máquina, ficheros fuente Pascal, etc., de forma que en todo momento una lectura del directorio por pantalla sólo muestre los ficheros que sean útiles en este instante, mejorando la presentación.

El formato de los nombres de fichero utilizados por AMSDOS es el mismo que en CP/M:

<nombre de fichero>. <extensión>

donde «nombre de fichero» es uncampo de ocho caracteres como máximo y «extensión» de tres, también como máximo. Los símbolos comodín (''*'' y ''?'') son tratados de forma análoga a como lo son en CP/M.

Aparte de los comandos específicos del AMSDOS, se encuentran los comandos propios del intérprete; éstos no van precedidos por la barra vertical y se utilizan para la carga en memoria y grabación de programas: SAVE, LOAD, MERGE, etc.

El volcado de un programa de memoria a disco o cinta se realiza mediante la orden SAVE. Por ejemplo:

```
SAVE "unfich"
```

hará que, en el caso de utilizar el disco como dispositivo de destino, aparezca en el mismo un fichero, denominado ''UNFICH.BAS'' (la extensión la pone automáticamente el sistema), conteniendo el programa BASIC que en ese momento reside en memoria. Si la orden fuera

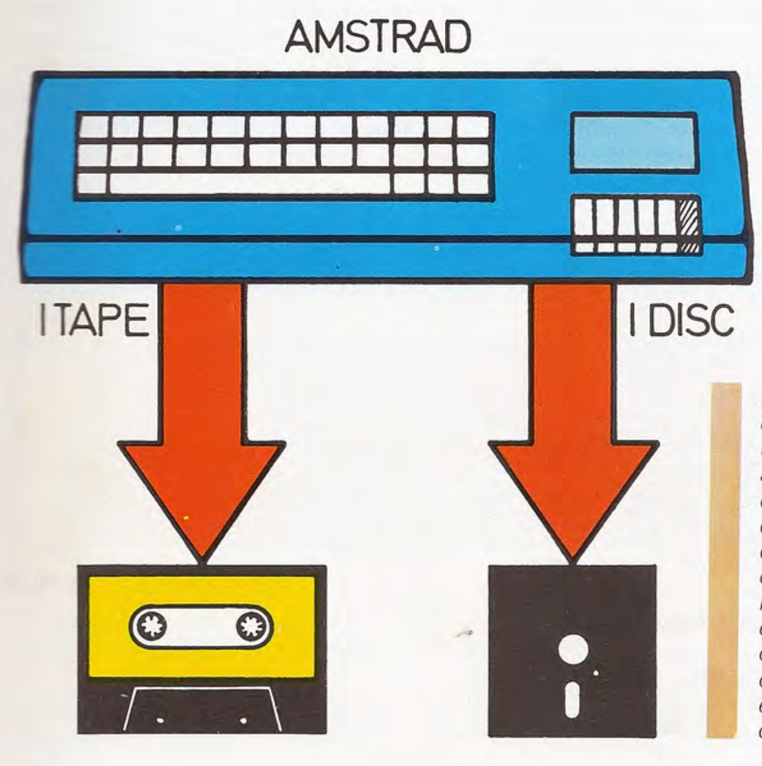
```
SAVE "unfich",P
```

el fichero quedaría protegido, de tal forma que sólo sería posible su ejecución, sin posibilidad de obtener un listado.

Al ejecutar la orden:

```
SAVE "unfich",A
```

el fichero se graba en modo ASCII; es decir, tal y como se ve al sacar un listado.




*La existencia de dos tipos de unidades de almacenamiento externo obliga a disponer de comandos que especifiquen hacia cuál de ellas deben dirigirse las operaciones de entrada/salida de información.*

Para grabar una zona de memoria en general (que puede ser el contenido de la pantalla o un bloque de código máquina) el parámetro a utilizar es "B"; en este caso también habría que especificar el comienzo y la longitud de la zona de memoria a volcar. Por ejemplo:

```
SAVE "pantalla",B,&C000,&4000
```

crea en disco el fichero "PANTALLA.BIN" (de nuevo, la extensión es adjudicada directamente al detectar el parámetro "B") que contiene información sobre la pantalla visualizada en ese momento.




*Pese a que la existencia de un directorio en disco hace innecesario que los ficheros cuyo soporte sea este medio incluyan una cabecera, AMSDOS la inlcuye con el fin de guardar la compatibilidad con los ficheros almacenados en cinta.*

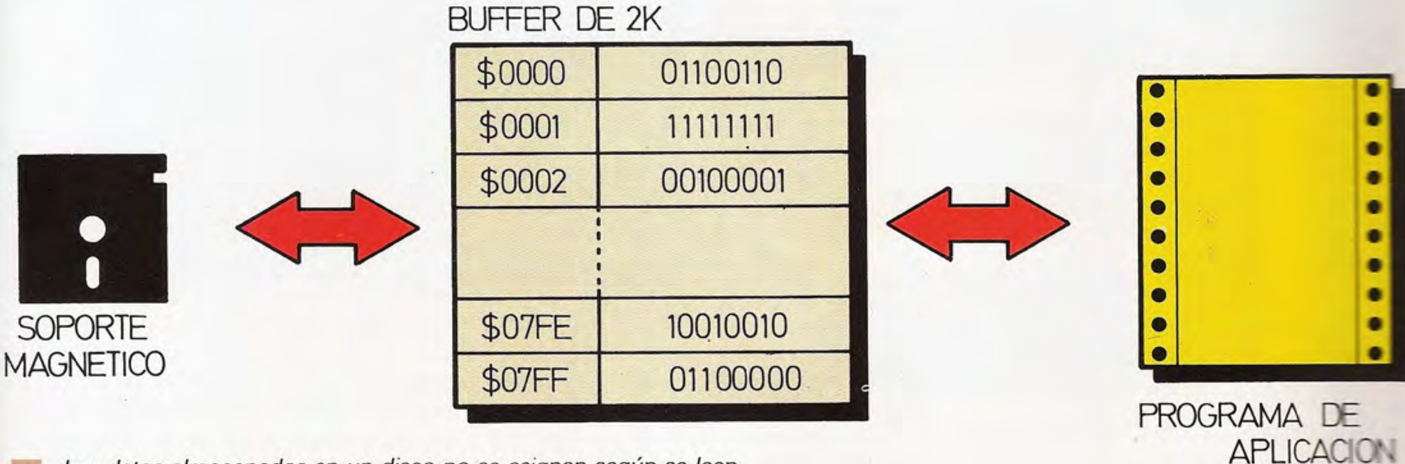
## CP/M, la estrella del conjunto

Sin duda, lo que hace que los ordenadores Amstrad se encuentren en la frontera entre el juguete y la utilidad práctica es la posibilidad de trabajar en CP/M; sistema operativo que ofrece un entorno ideal para los que velan sus primeras armas en informática, e incluso para la ejecución de aplicaciones profesionales.

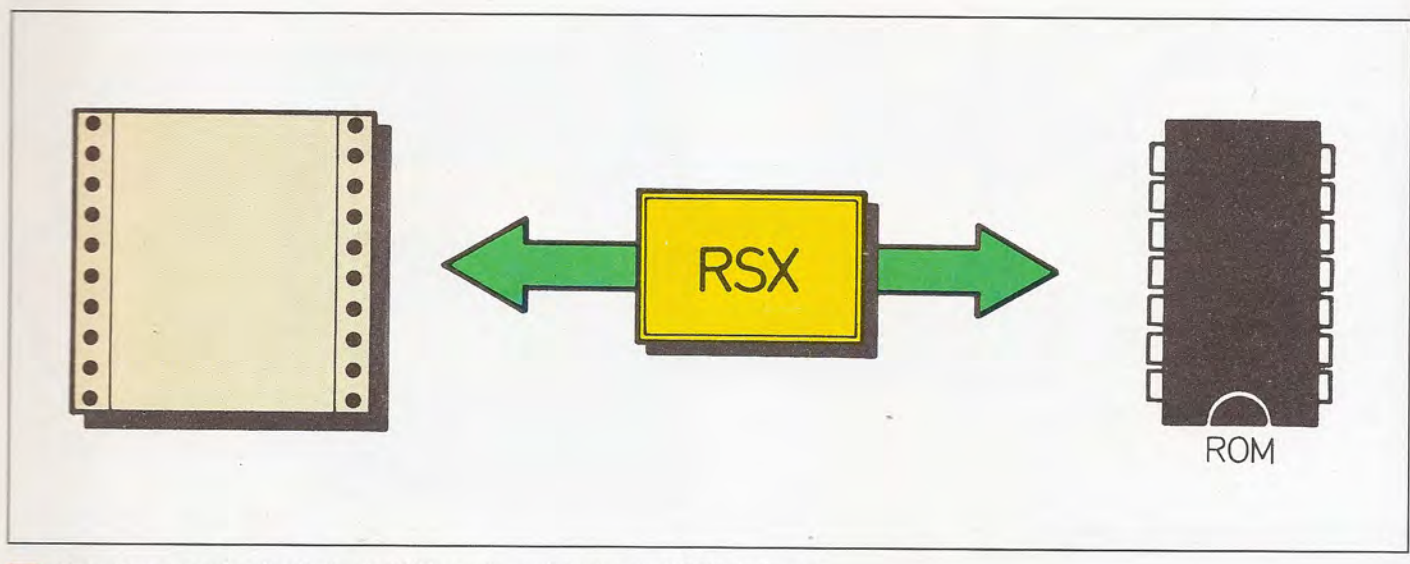
Tanto el CPC464 como el CPC664 operan con la versión 2.2 de este sistema operativo, con el serio inconveniente de la escasa memoria RAM que queda para los programas de usuario, concretamente 44K. Esto no es óbice para que no se puedan emplear excelentes programas: Wordstar, o los compiladores de Fortran y Cobol de Microsoft son tres buenos ejemplos.

Esta limitación de la memoria RAM disponible bajo CP/M queda resuelta en el modelo CPC6128, que incorpora 128K de RAM, el doble que sus predecesores, además de la versión 3.0 de CP/M. Esta última incorpora algunos comandos nuevos, así como un paquete destinado a aprovechar las capacidades gráficas del aparato desde el CP/M; los CPC464 y CPC664 no pueden hacer uso de esta posibilidad.

La compatibilidad entre los ficheros que manejan AMSDOS y CP/M sólo se mantiene a nivel de ficheros de texto. En consecuencia, estando bajo el control del AMSDOS, no se puede realizar un LOAD de un fichero .COM, aunque sí se pueden manipular los ficheros de texto que hubieran sido creados por un editor desde CP/M, por ejemplo. Ello se debe a que los ficheros generados por los programas almacenados con el comando SAVE sin especificar la opción "A" desde AMSDOS, van precedidos por una




*Los datos almacenados en un disco no se asignan según se leen, sino que se almacenan en una zona intermedia de memoria hasta que son necesarios.*




*Una vez que se ha diseñado un RSX puede «enlatarse» en ROM, de forma que esté siempre disponible nada más conectar el equipo.*

cabecera en la que se incluyen diversas características del fichero: tipo (BASIC, binario, ASCII, etc.), protección, longitud, zona de carga... mientras que los ficheros en CP/M no llevan este tipo de información. Por esta razón, al intentar cargar desde el AMSDOS un fichero generado por CP/M y detectar la ausencia de cabecera, se produce un error.

Como se ha comentado, los ficheros grabados con la opción "A" no incorporan dicha cabecera, por lo que pueden ser utilizados sin problemas en el entorno CP/M.

Entre los comandos CP/M que se encuentran en el disco de sistema hay algunos orientados a facilitar el uso de este sistema operativo con los ordenadores Amstrad. Por ejemplo, destacan CSAVE y CLOAD, para transferir contenidos de ficheros de cinta a disco y viceversa, y FILECOPY para transferir ficheros de un disco a otro sin necesidad de disponer de una segunda unidad. Entre los ausentes cabe mencionar SAVE, que se utiliza para guardar páginas de memoria en disco, y que puede ser de utilidad a los programadores avanzados en CP/M.

Para pasar al AMSDOS desde el CP/M, hay que ejecutar un pequeño programa llamado AMSDOS.COM que devuelve el control al intérprete Basic. Otra posibilidad alternativa es efectuar un reset del aparato, presionando simultáneamente las teclas CTRL, SHIFT y ESC. Si se desea pasar a CP/M desde el control del AMSDOS, hay que teclear la orden:

```
|CPM
```

## Gestión de ficheros

Todos los comandos de AMSDOS son interpretados desde el Locomotive Basic, lo cual permite al programador operar con ellos desde el propio intérprete (ver ejemplo de redefinición de comandos en el programa 1).

El usuario puede crear y gestionar sus propios ficheros desde el Basic. Para ello, antes de utilizarlos debe abrirlos. Una vez que haya terminado su manipulación tendrá que cerrarlos. El Locomotive Basic impone la restricción de que sólo puede estar abierto un fichero para lectura y otro para escritura en cada instante; además, todos los ficheros serán de acceso secuencial.

Para gestionar un fichero, ya sea de entrada o de salida, se reservan 2K de memoria RAM que actuarán como buffer entre el alojamiento final en RAM y el soporte magnético que representa el disco o la cinta.

El programa 2 incluye un ejemplo en el que se ha abierto un fichero para almacenar el contenido del array "a$". El primero consiste en crear un fichero con la orden OPENOUT, seguida del nombre y la extensión de tal fichero.

Hay que destacar que el nombre que aparecerá al listar el directorio del disco será exactamente el que figura entre las dobles comillas. Por lo tanto, si no da una extensión al fichero, como sucedería de ejecutar:

```
OPENOUT "PRUEBA"
```

en el directorio no aparecerá extensión alguna para el mismo. Lo contrario ocurre al utilizar el comando SAVE, con el que las extensiones se asignan automáticamente. En el Locomotive Basic las órdenes básicas de entrada/salida —fundamentalmente PRINT e INPUT— pueden ir seguidas por un número que especifica el canal por el que se realizará la operación. Concretamente, en el programa adjunto el canal asignado es el número nueve, que corresponde a la vía de comunicación con el sistema de disco/cinta.

Una vez que se ha enviado toda la información al fichero de destino, hay que

cerrarlo con la orden CLOSEOUT. El hecho de que no haya que especificar ahora el fichero que se cierra se debe a que, como se ha comentado unas líneas más arriba, sólo es posible tener abierto un fichero para salida; de ahí que no exista duda alguna sobre el fichero que hay que cerrar. Esta operación, realizada sobre el disco podría haber tenido lugar sobre la cinta si, con anterioridad a la ejecución de este segmento de programa, se hubiera ejecutado la orden:

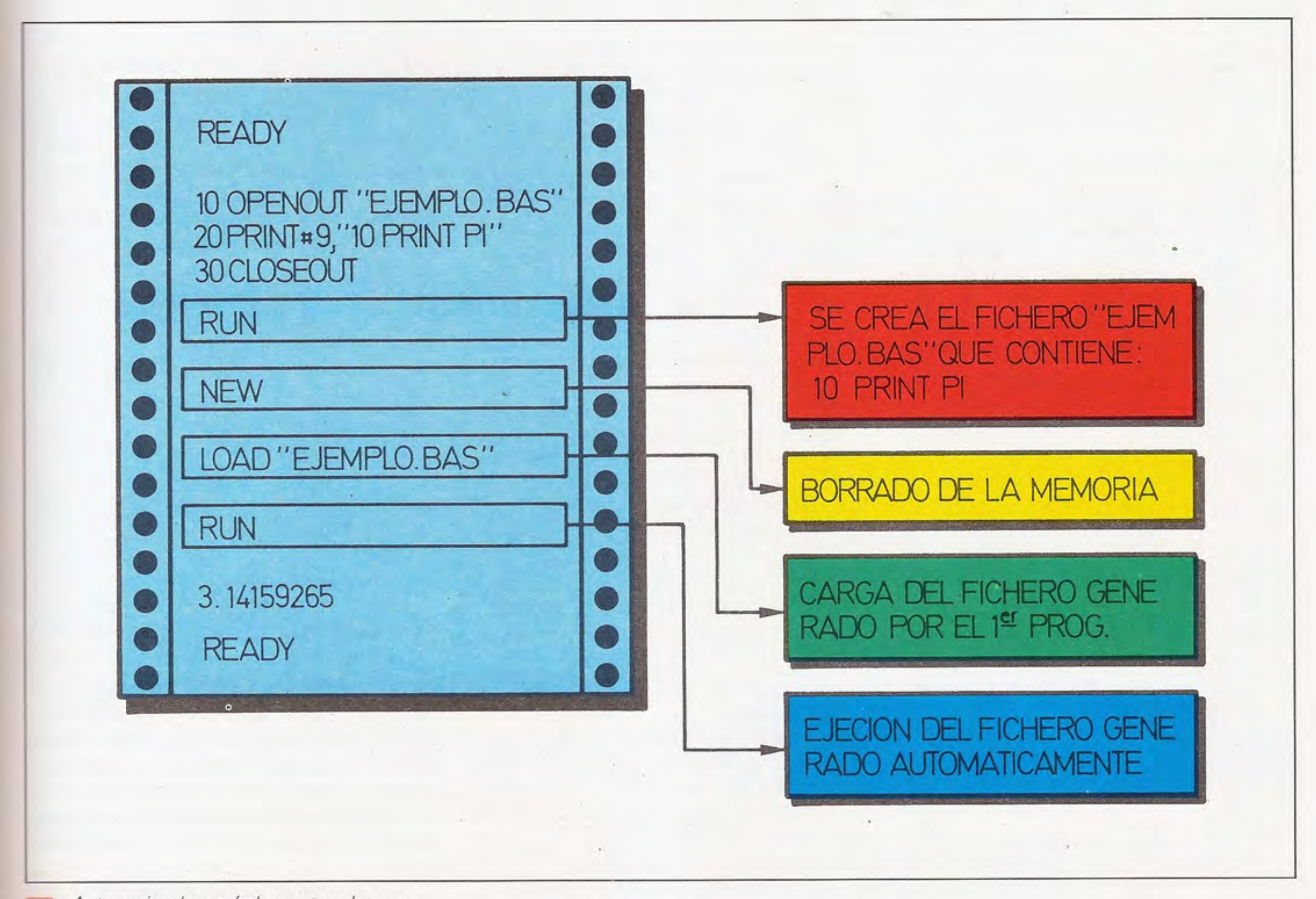
```
|TAPE
ó
|TAPE.OUT
```

bien incluyéndola como línea del programa, o bien tecleándola directamente con anterioridad a la ejecución del mismo. Cuando se conecta el ordenador por primera vez, toda la entrada/salida a través del canal número nueve está orientada hacia el disco.

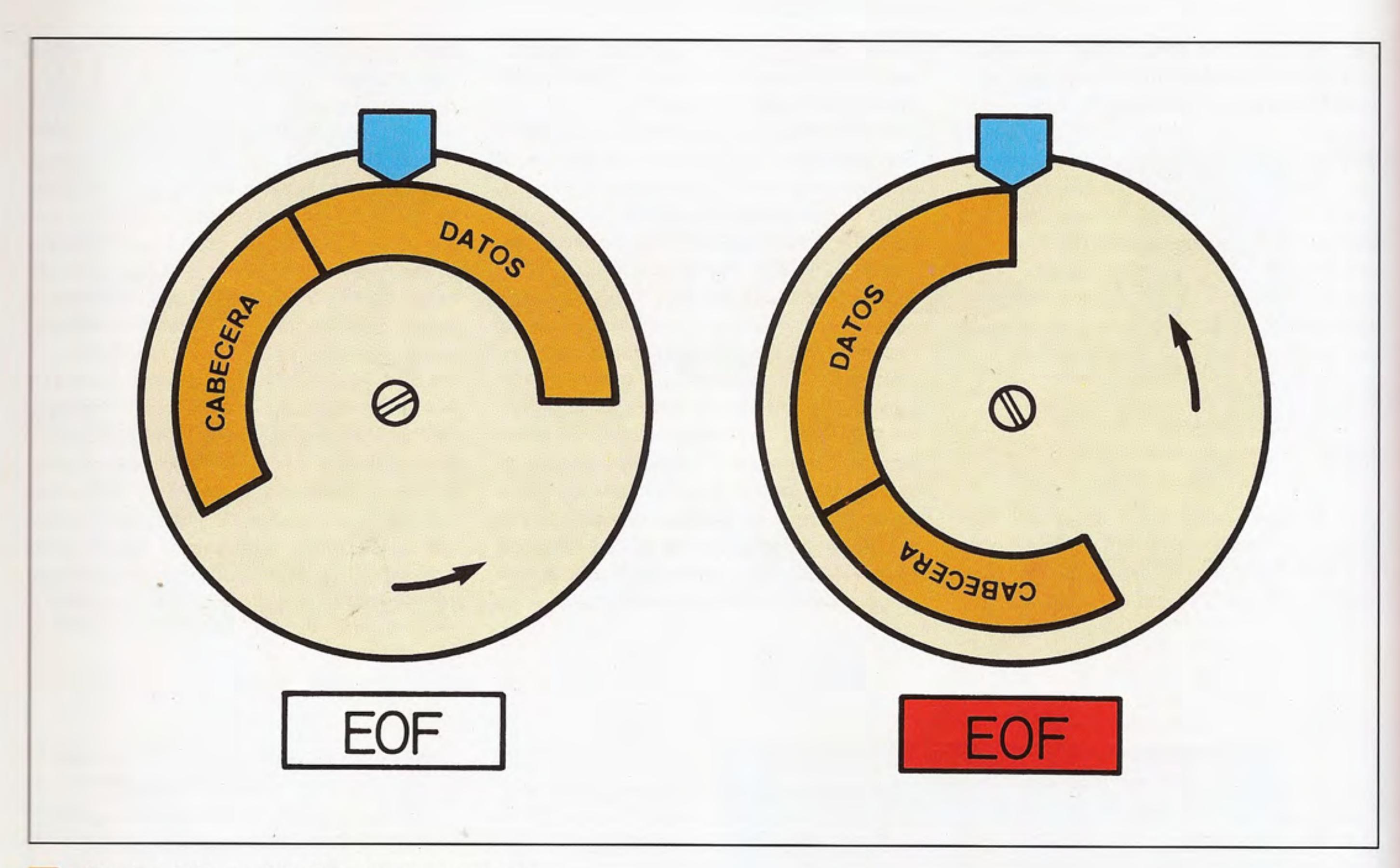
Una vez que se han grabado los datos sobre el disco, lo más normal es pensar en recuperarlos; al respecto, el programa 3 ofrece una alternativa. En él se observa la forma de abrir un fichero para lectura (OPENIN), para después cerrarlo con CLOSEIN. De nuevo, es innecesario especificar el nombre del fichero que se cierra por la restricción ya comentada.

Existe una alternativa más potente que la reflejada en la rutina en cuestión para la tarea de recoger datos del disco o cinta. En realidad, habrá pocas ocasiones en las que el programador conozca de antemano el número de datos que contiene el fichero que quiere leer. En tales situaciones, sería deseable disponer de algún medio que informara si se ha llegado al final del mismo e ir leyendo los datos mientras realmente se puedean ir cogiendo. Al efecto se dispone de la variable EOF, la cual es cierta cuando se llega al final del fichero abierto para lectura. En el programa 4 se muestra el uso típico de esta variable. El empleo de la estructura WHILE-WEND acompañando a la variable EOF es un buen ejemplo de matrimonio perfecto, copiado de otros lenguajes más evolucionados como son C o Pascal.

Los ficheros creados pueden guardar cualquier tipo de información, ya sea numérica o literal, quedando bajo la responsabilidad de quien posteriormente utilice el fichero al recoger la información de la misma forma en la que fue depositada. Como se obbserva en los programas 5 y 6, la información se graba y se recupera en forma de parejas (dato-




*Actuación de un fichero creado por medio de la orden OPENOUT.*




*El valor de la variable EOF permite reconocer el momento en que se llega al final del fichero.*

**Principales comandos del AMSDOS**

| RSH | Acción | Ejemplo |
|---|---|---|
| \|DIR | Listado por pantalla del directorio del disco. | \|DIR,"*.BAS" |
| \|A | La unidad de disco implícito será la "A". | \|A |
| \|B | Como el anterior para la unidad "B". | \|B |
| \|CPM | Pasa el control al CP/M. | \|CPM |
| \|ERA | Borrado de ficheros. | \|ERA,"PRUEBA.*" |
| \|REN | Cambio del nombre de un fichero. | \|REN,"NUEVO.BAS","VIEJO.BAS" |
| \|USER | Definición de un área de usuario. | \|USER,5 |

| RSH | Acción |
|---|---|
| \|DISC.IN | Indica que las operaciones de lectura se harán sobre el disco. |
| \|DISC.OUT | Especifica que las operaciones de escritura se realizarán sobre el disco. |
| \|DISC | Señala que las operaciones de lectura y escritura se realizarán sobre disco. |
| \|TAPE.IN | Indica que las operaciones de lectura se harán con la cinta. |
| \|TAPE.OUT | Señala que las operaciones de escritura se harán con la cinta. |
| \|TAPE | Tanto las operaciones de lectura como de escritura se realizarán sobre cinta. |

literal, dato-numérico). Un intento de realizar la entrada de los datos en el programa a través de la orden:

```
INPUT#9,b(i),a$(i)
```

puede traer malas consecuencias para la correcta ejecución del programa.

## Una aplicación: Basic en español

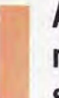
Los ficheros creados a partir de la orden OPENOUT tienen la particularidad de que, en el caso de que su contenido sean programas Basic, es posible cargarlos en memoria y ejecutarlos como si realmente hubieran sido tecleados desde el propio editor del Locomotive Basic.

En el cuadro final se muestra un ejemplo claramente ilustrativo. En un

primer paso se crea el fichero "EJEMPLO.BAS" a través de OPENOUT y CLOSEOUT cuyo contenido es:

```
10 PRINT PI
```

Esto es justamente una línea de programa Basic que, en el caso de aparecer en un programa, imprimiría por pantalla el valor 3.14159265. Por lo tanto, lo que contiene el fichero recién creado es un pequeño programa Basic que escribirá dicho valor por pantalla. A continuación, podemos cargar el fichero "EJEMPLO.BAS" y ejecutarlo, obteniendo el efecto deseado.



## Almacenamiento en memoria de variables suscritas

Un conjunto de elementos, como ya se ha comentado, puede almacenarse en memoria bajo un mismo nombre de variable suscrita. Para distinguir a un elemento de otro es necesario recurrir a una serie de números, separados por comas, que reciben el nombre de «índices». Estos acompañan al nombre de la variable e identifican a cada uno de los elementos.
Cuando se define una estructura de este tipo, se indica el número de elementos distintos que van a residir en memoria bajo el mismo nombre genérico. En general, el acceso a cada posición de memoria se realiza a través de un número que identifica su emplazamiento o «dirección» dentro del espacio de memoria. Cuando el ordenador desea conocer el valor de una variable, le basta con examinar el contenido de la posición de memoria asociada a dicha variable; dependiendo del tipo de variable de que se trate, leerá, a partir de esa posición, el contenido de otras posiciones contiguas. En el caso de los conjuntos de variables o «arrays», la cosa no es tan sencilla. La existencia de más de un elemento bajo el mismo nombre genérico crea alguna dificultad. Si bien, es evidente que a partir del nombre de la variable y del valor de los índices, será posible acceder a cualquiera de los elementos del conjunto. La forma de acceder a los elementos de las variables suscritas no presenta excesivas complicaciones. En primer lugar, consideremos un conjunto de tan sólo una dirección. La variable del conjunto o «array» tendrá asociada una dirección de memoria; dirección a partir de la que estará localizada su zona de almacenamiento. En segundo lugar, hay que precisar si nuestro ordenador comienza a numerar los elementos del «array» a partir del índice 0 ó del 1. Es posible que quepa la elección mediante el comando OPTION BASE. En cualquier caso, en el ejemplo se considerará que la numeración empieza con el índice cero. Supondremos, asimismo, que i es el índice que corresponde a un determinado elemento del «array», que L es el tamaño en bytes de cada uno de los elementos y que D es la dirección de memoria asociada a la variable de conjunto. La representación gráfica de su almacenamiento es la que refleja el gráfico adjunto (figura a). Las direcciones de memoria que interesa son:

**D: para el primer elemento, de índice 0**
**D+L: para el segundo elemento, de índice 1**
**D+2L: para el tercer elemento, de índice 2**
**D+3L: para el cuarto elemento, de índice 3**

Sin mayores dificultades, es posible obtener una ecuación que permita calcular las direcciones de cada elemento:

**DIR=D+(L*i)**

En el supuesto de que el índice no empezara a partir de 0, sino que lo hiciera a partir del valor 1, bastaría con un simple retoque para que la ecuación permitiera obtener la dirección efectiva de cada elemento:

**DIR=D+(L*(i—1))**

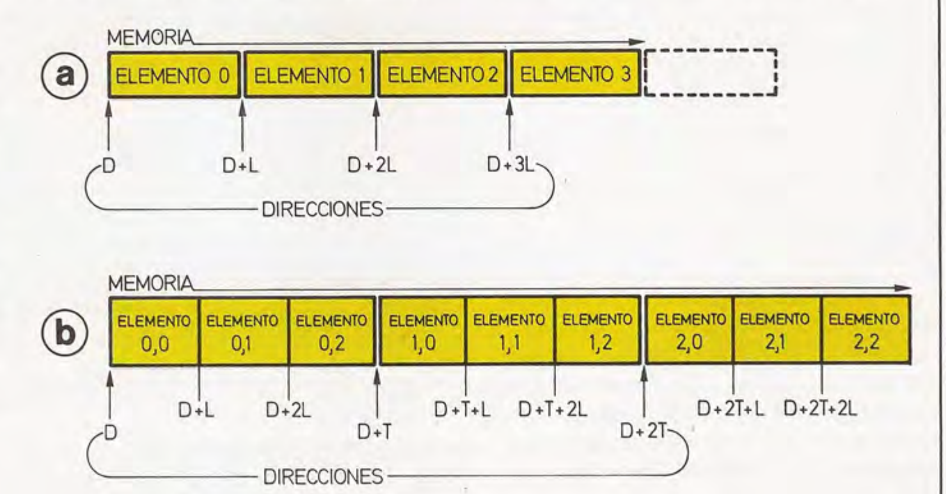
Tal como se observa, el cálculo de la dirección parte de dos componentes. Por un lado, una dirección absoluta (D), coincidente con la dirección inicial de la zona de almacenamiento reservada a la variable suscrita; y por otro, de un valor relativo que indica la distancia al elemento desde el comienzo de la zona de almacenamiento reservada: L*(i-1). En efecto, este valor depende del índice propio de cada elemento del «array». ¿Qué sucede cuando se tiene más de una dimensión? La complicación no es excesiva. Se trata de añadir un nuevo valor de desplazamiento relativo a la ecuación. Imagine que se trata de un libro (la variable suscrita), cuyo primer índice indica el número de página, mientras que el segundo identifica a cada palabra de la misma. Para localizar la posición de una determinada palabra, es necesario conocer a cuantas palabras de distancia del comienzo de la página se encuentra. Veamos, por ejemplo, cómo se almacena en la memoria un «array» coincidente con una matriz de dos dimensiones (índices i y j) con 3×3 elementos (ver figura b). En esta ocasión, T es el tamaño del subconjunto de todos los elementos que tienen el primer índice común. En consecuencia, si el número de elementos para ese índice es N, el valor de T será igual a N*L.
El cálculo de la dirección de comienzo de cualquier elemento se reduce, pues, a la siguiente expresión:

**DIR=D+(i*T)+(j*L)**

Aplicado al ejemplo de la figura b, se observa que, en efecto, la posición de memoria ocupada por el elemento 1,2 coincide con:

**DIR=D+(1*T)+(2*L)=D+T+2L**

En el cálculo de la dirección efectiva de cada elemento intervienen, pues, tres sumandos. El primero (D) coincide con la dirección absoluta a partir de la que comienza la zona de almacenamiento del «array». El segundo es un desplazamiento relativo al primer índice, que nos coloca al principio de la porción de memoria ocupada por los elementos que tienen el primer índice común. Por último, el tercer sumando es el que nos lleva, definitivamente, a la dirección del elemento específico; en efecto, equivale a un desplazamiento relativo al valor del segundo índice.
Este método es extensivo para el cálculo de las posiciones en las que se almacenan los elementos de los «arrays» de cualquier número de dimensiones.



**PROGRAMA 1**

```
10 INPUT "Comando ", a$
20 IF a$="directorio" THEN |DIR;"*.*"
30 IF a$="a cpm" THEN |CPM
40 GOTO 10
```

**PROGRAMA 2**

```
DIM a$(10)
...
...
...
OPENOUT "PRUEBA.DAT"
FOR i=1 TO 10
   PRINT #9, a$(i)
NEXT i
CLOSEOUT
```

El resultado es análogo al que se obtendría si dicho programa se hubiera cargado en memoria por medio del editor, y luego se hubiera pasado a disco ejecutando un SAVE.

Todo esto permite pensar en un programa capaz de leer cadenas cuyo contenido sea un programa Basic en español, y de generar un fichero .BAS con el equivalente en auténtico Basic. Posteriormente, este fichero puede ser cargado en memoria y ejecutado por el intérprete. Un array de strings (lin$) contendría el programa fuente de forma parecida a:

```
lin$(1)="10 PARA I=1 HASTA 10"
lin$(2)="20 ESCRIBE I"
lin$(3)="30 SIGUIENTE I"
```

El «compilador» de Basic-español a Basic-Real debería analizar los contenidos de lin$(1), lin$(2), etc., y generar sus equivalentes en inglés, para después grabarlos en disco como sigue:

```
PRINT #9,"10 FOR I=1 TO 10"
PRINT #9,"20 PRINT I"
PRINT #9,"30 NEXT I"
```

El fichero creado de esta forma puede ahora ser cargado en memoria y ejecutado como si de un programa Basic normal se tratara; con la particularidad de que su generación se ha realizado de forma automática, sin necesidad de haber pasado previamente por el editor del intérprete Basic.

**PROGRAMA 3**

```
OPENIN "PRUEBA.DAT"
FOR i=1 TO 10
   INPUT #9, a$ (i)
NEXT i
CLOSEIN
```

**PROGRAMA 4**

```
OPENIN "PRUEBA.DAT"
i=1
WHILE NOT EOF
     INPUT #9, a$(i)
     i=i+1
WEND
CLOSEIN
```

**PROGRAMA 5**

```
DIM a$(5), b(5)
...
...
...
OPENOUT "DATOS.DDD"
FOR i=1 TO 5
   PRINT #9, a$(i), b(i)
NEXT i
CLOSEOUT
```

**PROGRAMA 6**

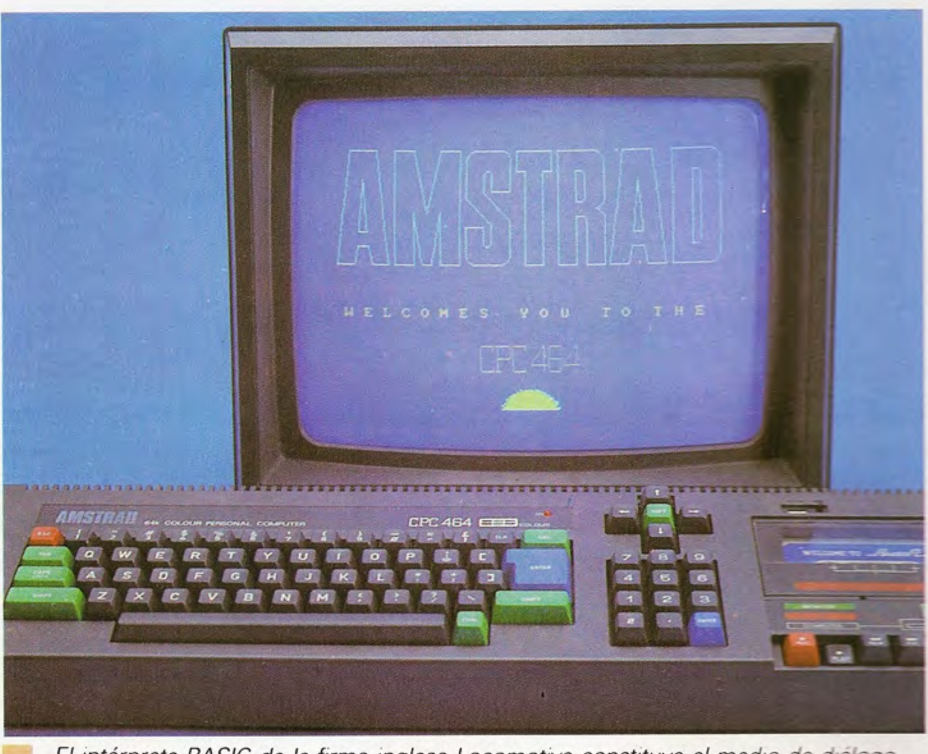
```
OPENIN "DATOS.DDD"
i=1
WHILE NOT EOF
     INPUT #9, a$(i), b(i)
     i=i+1
WEND
CLOSEIN
```

# Locomotive BASIC

## El dialecto BASIC de los ordenadores Amstrad



Los ordenadores Amstrad (CPC-464, CPC-664 y CPC-6128) están todos ellos dotados de intérpretes BASIC creados por la firma inglesa Locomotive. Como ya es habitual en el mundo de los intérpretes de este popular y omnipresente lenguaje de programación, cada dialecto incorpora ciertas características que lo distinguen. A través de los próximos párrafos se describirán las particularidades más significativas del Locomotive BASIC; algunas tan notables como la posibilidad de definir ventanas en la pantalla y gestionar las interrupciones bajo el control del propio intérprete BASIC.



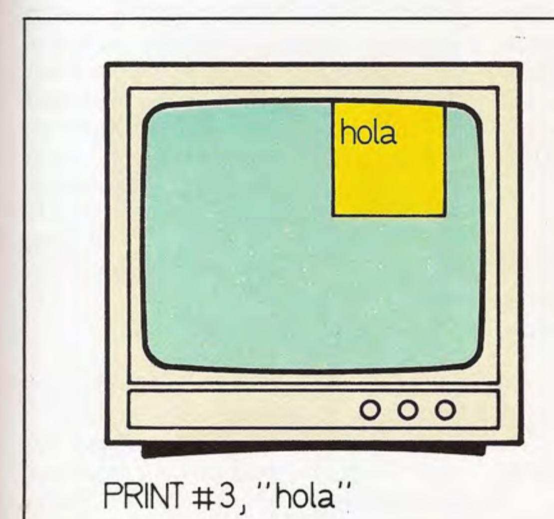

*El intérprete BASIC de la firma inglesa Locomotive constituye el medio de diálogo con los ordenadores Amstrad. La fotografía muestra el modelo CPC-464.*

### La pantalla

El usuario puede trabajar con tres formatos diferentes de pantalla o «modos», de los que sólo uno de ellos estará activo en cada momento. Los modos 0, 1 y 2 se corresponden con pantallas de 20, 40 y 80 columnas respectivamente. En todos los modos de pantalla el número de líneas horizontales es de veinticinco.

Se pueden manejar hasta 27 colores, aunque el número de ellos visualizables en cada momento depende del modo de definición de la pantalla: a mayor definición, menor número de colores disponibles para visualizar, ya que el ordenador debe almacenar en un espacio de 16K toda la información relativa a la pantalla. A mayor número de caracteres imprimibles, menor es el espacio que queda para guardar la información sobre el color.

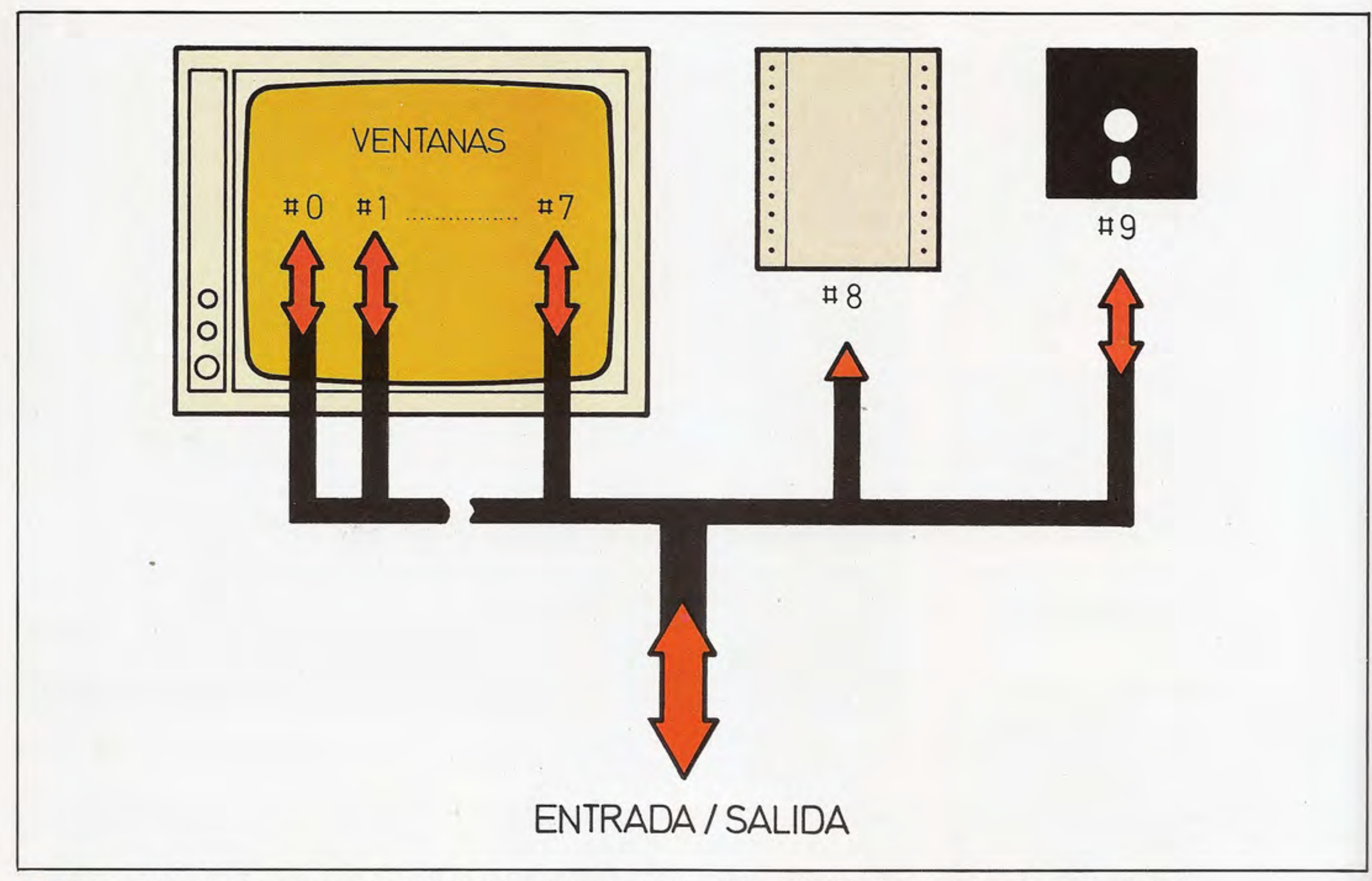
Siempre se dispone de quince plumas con las que escribir los caracteres, elegibles mediante una instrucción PEN, a las cuales puede asociarse la tinta que el usuario desee a través de la instrucción INK. Los colores del papel sobre el que se escribe y del borde de la pantalla son también definibles, pudiendo incluso hacer que sean parpadeantes y variar el período de parpadeo.







*Las ventanas son una potente herramienta que Locomotive pone a disposición del programador de aplicaciones*




*Los canales constituyen una extensión del concepto de ventana a los periféricos de escritura y almacenamiento masivo. Así, el canal número 8 corresponde a la impresora, y el 9 a la unidad de disco.*

Aparte de las posibilidades cromáticas, disponibles en la gran mayoría de los ordenadores personales, los ordenadores Amstrad poseen otra facultad no tan generalizada; ésta es la de definir y gestionar hasta ocho ventanas independientes de texto sobre el monitor o TV.

Una ventana es un espacio definido sobre la pantalla del monitor que el programador puede gobernar como si se tratara de otra «pequeña pantalla», definiendo los colores del papel y de los caracteres de forma independiente para cada una de ellas. La ventana número cero (#0) está asignada a todo el espacio físico de la pantalla del monitor.

Una pantalla se define a través de la instrucción WINDOW. Por ejemplo, la orden:

```
WINDOW #3,10,20,1,20
```

define la ventana número 3. Las cuatro últimas cifras del argumento indican el tamaño de la ventana: entre las líneas 10 y 20 y las columnas 1 y 20. Los comandos típicos de interacción con la pantalla, como PRINT, INPUT o LOCATE (para situar el cursor en un punto determinado de la misma), deben ir seguidos por un número indicativo de la pantalla sobre la que deba reflejarse su acción. Si no es así, la máquina da por supuesto que las órdenes en cuestión hacen referencia a la ventana cero. Por ejemplo:

```
PRINT #3, "HOLA"
```

escribirá la cadena de caracteres HOLA en la esquina superior derecha de la pantalla, dentro de la ventana anteriormente definida; mientras que:

```
PRINT "HOLA" ó PRINT #0, "HOLA"
```

lo hará en el espacio definido para toda la pantalla.

En realidad, las ventanas son la particularización a un caso concreto de un concepto más amplio: el de «canal» (ver figura).

Toda la actividad de entrada/salida, que se lleva a cabo fundamentalmente a través de PRINT e INPUT, se puede asignar a un canal concreto. Cuando el número de canal está comprendido entre 0 y 7, se trata de una ventana de texto sobre la pantalla. Si este número es el 8, la salida se dirigirá hacia la impresora y si es el 9 se corresponde con el canal de comunicación con la unidad de disco o cinta. Por lo tanto, para obtener un listado por impresora, la orden a ejecutar es:

```
LIST #8
```

la cual dirige el listado hacia el canal número ocho.

Las ventanas son una excelente herramienta para confeccionar programas de utilidad, en los que un factor decisivo es la facilidad que el usuario encuenre en su manejo: puede haber ventanas

dedicadas a la presentación de menús, otras que muestren el estado en el que se encuentra la ejecución del programa, y otras cuyas misión sea recoger las entradas que realiza el usuario a través del teclado.

También existe la posibilidad de trabajar en modo gráfico, con una resolución máxima de 640×400 pixels. Mediante los comandos adecuados se pueden elegir los colores de papel y de la pluma de gráficos.

Como viene siendo habitual, el Locomotive Basic también incluye todo un repertorio de comandos para dibujar líneas y puntos y mover el cursor gráfico a voluntad del programador. Aunque se echan de menos instrucciones típicas para el dibujo de círculos, si se dispone de comandos para dibujar líneas discontinuas (MASK) y para rellenar superficies delimitadas por líneas dibujadas con anterioridad (FILL); desgraciadamente, este último comando sólo está presente en el Basic del CPC 664.

## Los sonidos

Las posibilidades que en este campo ofrece el Locomotive Basic son enormes. En primer lugar se dispone de tres canales independientes de sonido, lo que da la posibilidad de conectar el ordenador a un amplificador estéreo para aprovechar al máximo sus características y mejorar la calidad ofrecida por el pequeño altavoz incorporado. Un canal se corresponde con el altavoz izquierdo, otro con el derecho y el tercero de ellos con ambos a la vez (ver figura).

Todos los sonidos se controlan a través de la instrucción SOUND, la cual admite hasta un total de siete parámetros. El primero de ellos especifica la situación de los canales anteriormente comentados; el segundo el período del tono emitido; el tercero su duración, expresada en centésimas de segundo con un valor comprendido entre 1 y 32767, y el cuarto especifica el volumen con un valor entre 0 y 15. Si este último no se define, el ordenador tomará el valor 12 por defecto.

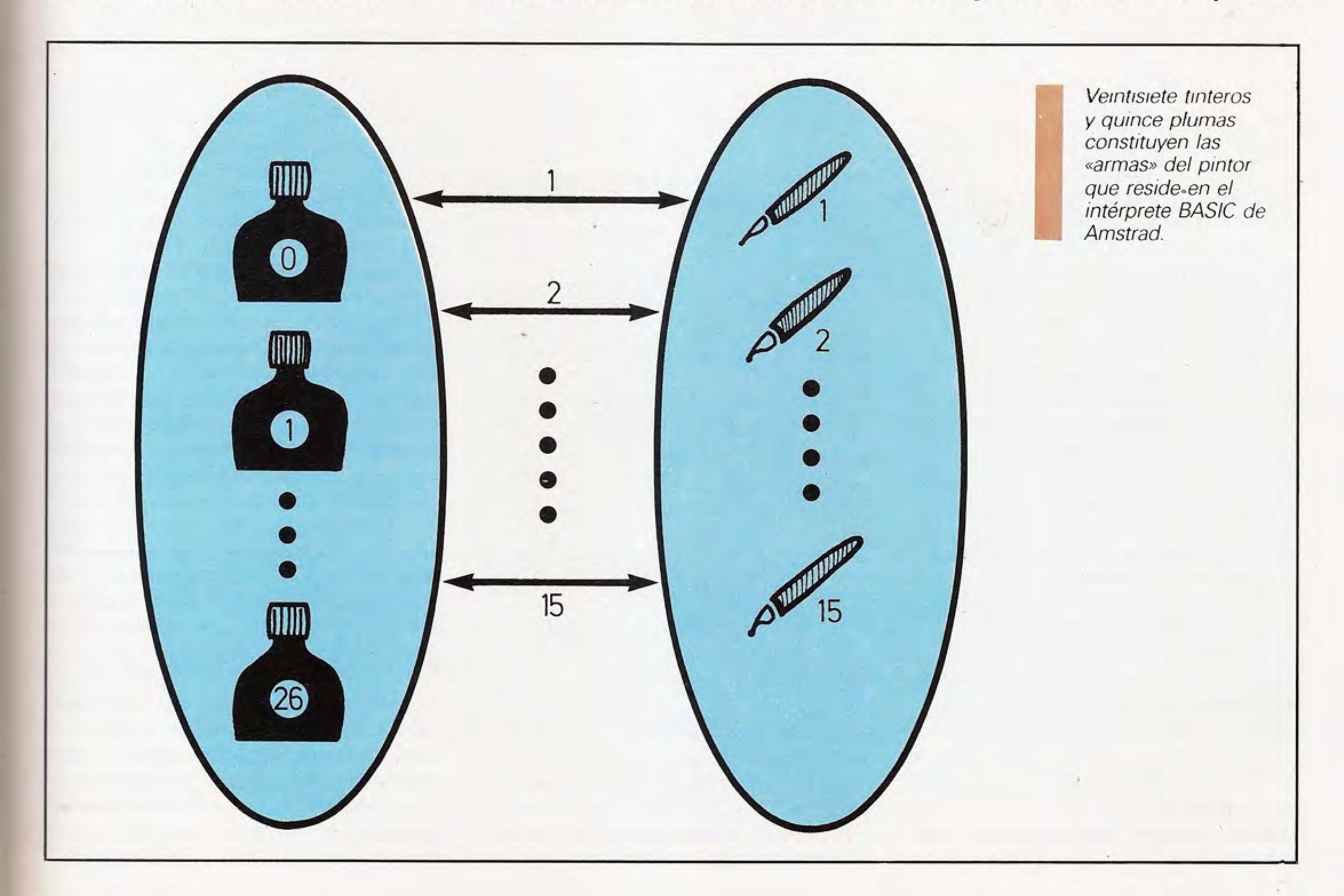
Cada canal tiene asociado una cola que puede contener hasta cinco sonidos en espera de ser ejecutados. Suponga que tecleamos la siguiente orden seguida de una pulsación sobre ENTER:

```
SOUND 1,284,3000
```

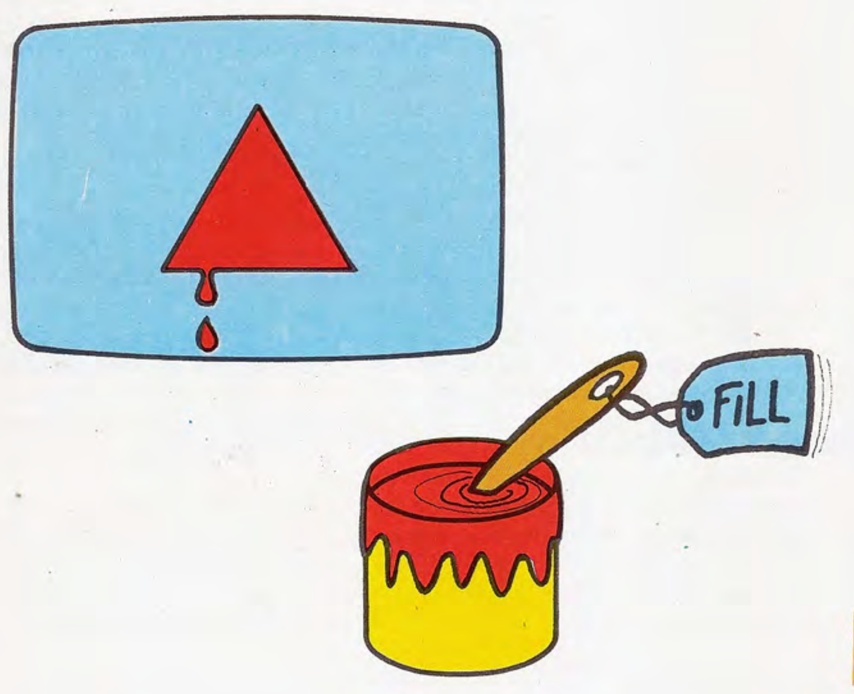
Esta hará sonar por el altavoz incorporado al equipo la nota LA durante unos cinco minutos. Mientras está sonando esta nota es posible operar normalmente con el ordenador ya que, al pulsar ENTER, la orden fue almacenada en la cola de sonido del canal 1 para que la gestionara el chip correspondiente, pudiendo incluso teclear

```
SOUND 1,478,200 <ENTER>
```

cuyo efecto es pasar a la cola de sonido del canal 1 un DO de duración igual a dos segundos. Al acabar la ejecución




*Veintisiete tinteros y quince plumas constituyen las «armas» del pintor que reside en el intérprete BASIC de Amstrad.*




*El comando FILL permite rellenar con la tinta especificada un área delimitada por líneas continuas.*

del LA, transcurridos los 5 minutos, el chip de sonido recogerá el DO de la cola y lo ejecutará.

Además se dispone de los comandos necesarios para sincronizar sonidos por los diversos canales disponibles, y controlar el estado de las colas para tomar decisiones en función de sus contenidos en la ejecución de programas.

Por si esto fuera poco, se pueden definir los contornos de las envolventes de volumen y de tono, con las que los sonidos generados dejan de ser fríos pitidos, a través de las instrucciones ENV y ENT respectivamente. De igual forma que una envolvente de volumen especifica la variación relativa de amplitud del sonido emitido en el tiempo, una envolvente de tono realiza la misma definición sobre el período del tono emitido.

Se pueden definir hasta quince envolventes de cada tipo, identificando a cada una por un número. Si el sonido generado por una instrucción SOUND va a hacer uso de alguna envolvente, el número de ésta se indica como 5.º parámetro para las de volumen y en el 6.º para las de tono.

Un ejemplo de envolvente de volumen puede ser:

```
ENV 1,5,3,4,5,-3,8
```

El primer parámetro especifica el número de la envolvente de volumen (1) por el que será referenciada en las instrucciones SOUND que la utilicen. Los siguientes parámetros hacen que la nota crezca en cinco etapas, aumentando el volumen tres unidades en cada una de ellas, siendo la duración de cada una de cuatro centésimas de segundo. A continuación, el volumen caerá en cinco etapas de tres unidades de volumen cada una, durando cada una 8 centésimas. Observemos que, puesto que una envolvente de volumen —y de tono también— especifica la duración de la nota, el correspondiente parámetro de la instrucción SOUND puede tomar el valor cero. Por ejemplo:

```
SOUND 1,284,0,0,1
```

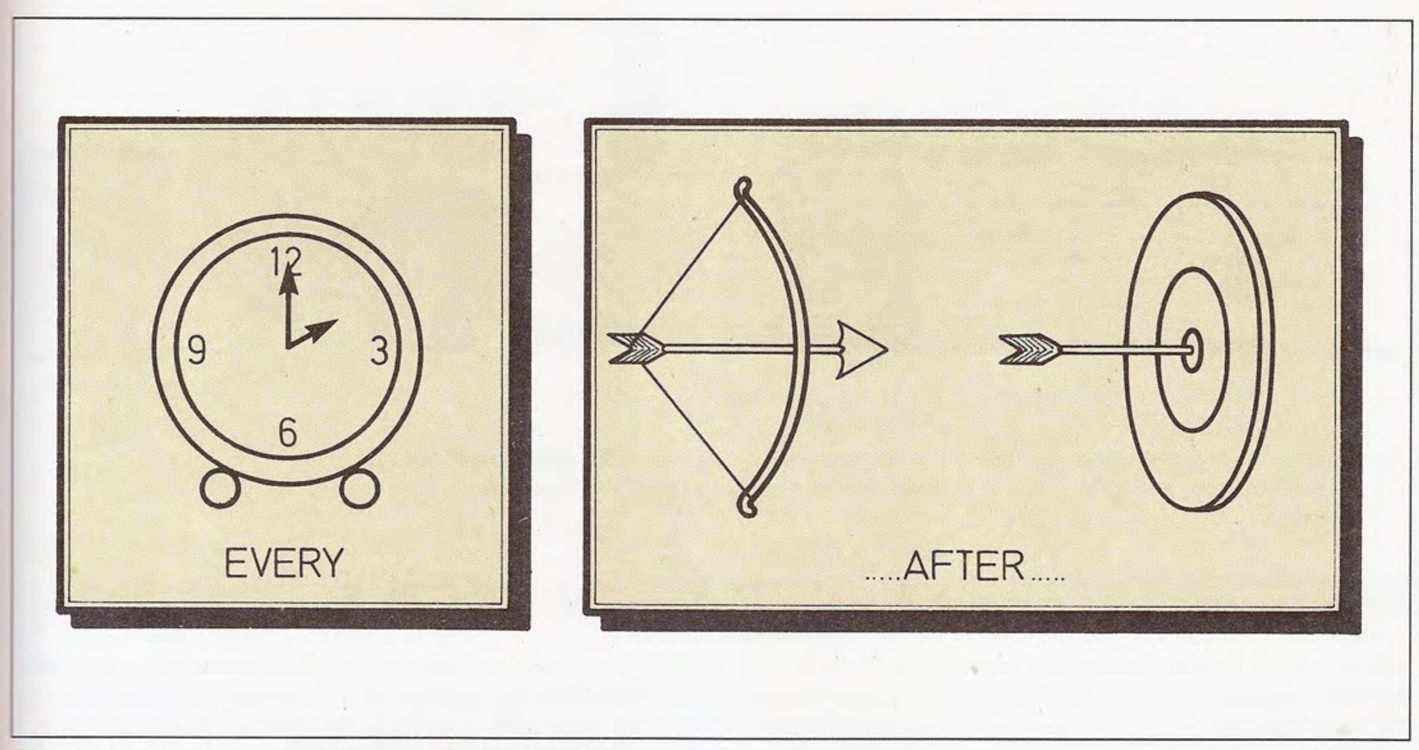
envía un LA por el canal 1, estando controladas su amplitud (que comienza en cero) y su duración por la envolvente de volumen número 1 (último parámetro).

Para finalizar la cuestión del sonido, comentar que, a base de variar el valor del parámetro de tono de la instrucción SOUND, se puede barrer ocho octavas musicales, y que su último parámetro se utiliza para añadir ruido a los tonos generados. Tal posibilidad resulta útil para imitar sonidos secos, como los producidos por instrumentos de percusión o por una locomotora de vapor. También cabe resaltar que la versatilidad de los comandos enunciados hace pensar en la posibilidad de sintentizar la voz. De hecho, existe un programa denominado «3-D Voice Chess» que consiste en un juego de ajedrez que brinda la posibilidad de que el ordenador recite sus jugadas a la vez que las ejecuta. El resultado es, sin embargo, bastante pobre debido a las dificultades que presenta la síntesis de voz humana.

## Interrupciones y temporizadores

En el Locomotive Basic encontramos una característica poco común entre los ordenadores de su rango: la posibilidad de gestionar las interrupciones desde el propio intérprete Basic.

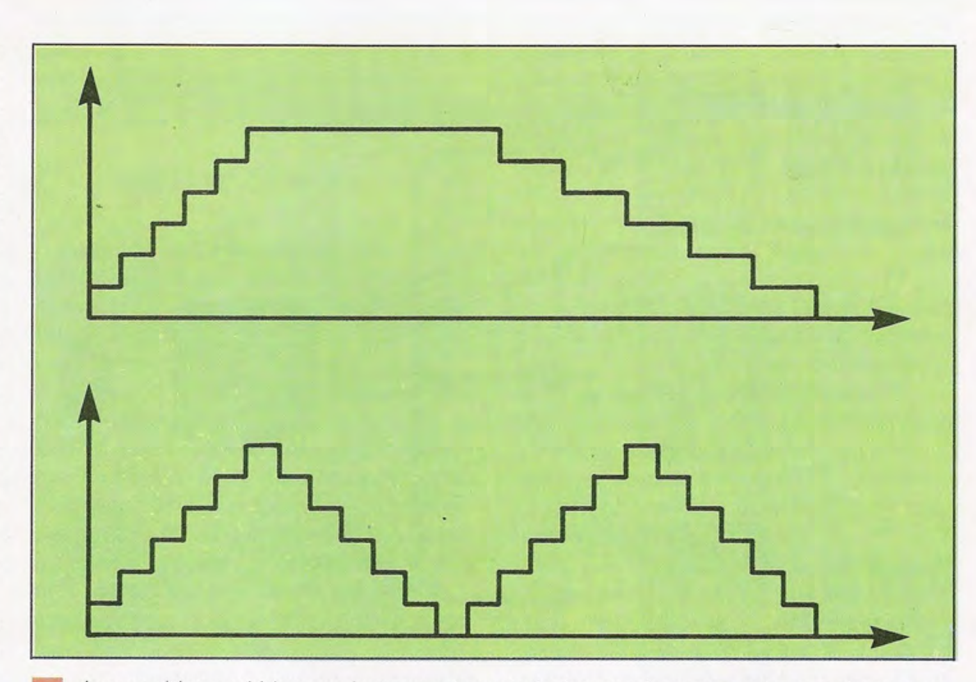
Una interrupción es un evento inter-



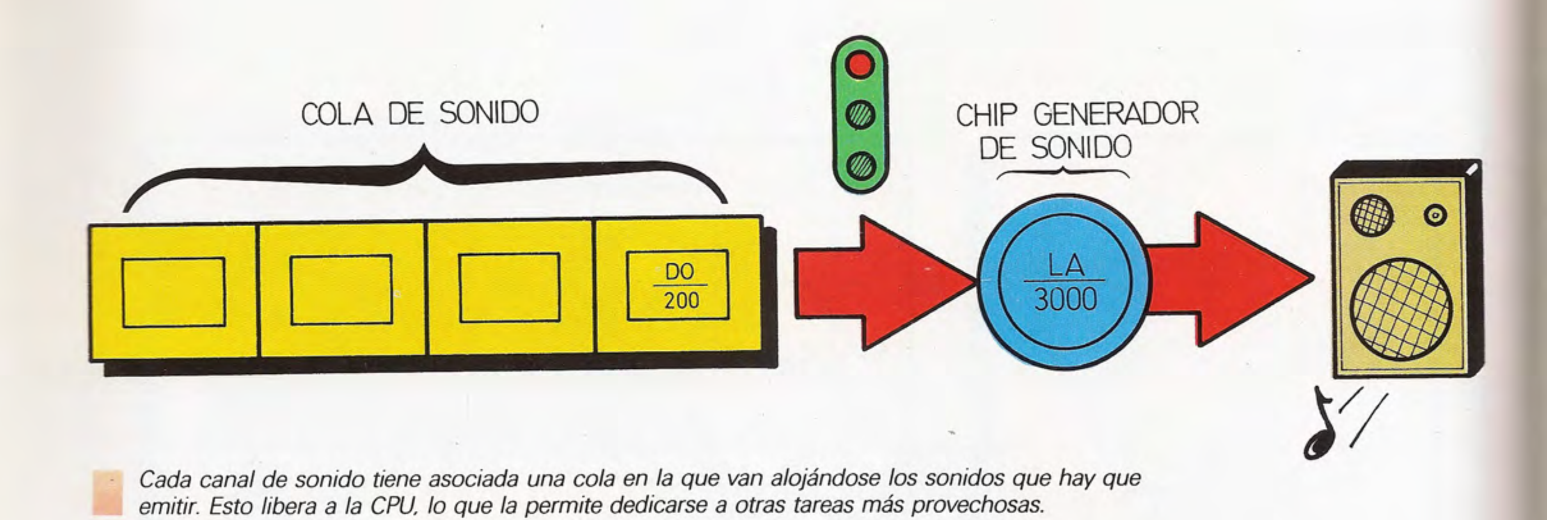

Una clara diferencia entre los comandos EVERY y AFTER.

no o externo a la máquina que obliga a suspender momentáneamente la ejecución del programa en curso, para atender a la causa que produjo dicha interrupción. Las interrupciones tienen distinta prioridad según la importancia de los eventos que las producen: no es lo mismo atender la interrupción que puede generar el operador al pulsar la tecla ESCAPE o ˆC al ver que su programa se ha metido en un bucle sin salida, que la que produce un hipotético circuito detector de fallo de la alimentación, en cuyo caso el interés primordial es intentar salvar en memoria permanente (disco por ejemplo) lo más que se pueda de lo que en ese momento estaba en memoria principal, relegando a un segundo plano a las restantes interrupciones (ver figura).

Existen cuatro temporizadores que permiten generar interrupciones en la ejecución del programa principal. Su acción consiste en bifurcar a una subrutina específica cada vez que se cumple el tiempo prefijado, volviendo de nuevo al programa principal una vez que la ruti-



*Los sonidos emitidos por instrumentos musicales son bastante más complejos que simples pitidos. Respecto de la amplitud, se caracterizan por un período de tiempo en el que ésta sube muy pronunciadamente, manteniéndose después constante para, a continuación, decaer con suavidad. La envolvente de volumen puede apreciarse también claramente en la figura (parte inferior).*




Cada canal de sonido tiene asociada una cola en la que van alojándose los sonidos que hay que emitir. Esto libera a la CPU, lo que la permite dedicarse a otras tareas más provechosas.

na seleccionada ha finalizado con un RETURN. Además, existe una quinta manera de ocasionar una interrupción: realizando un «Break», lo que se consigue pulsando la tecla de ESCape dos veces consecutivas.

El orden de prioridad de las interrupciones es, de mayor a menor:

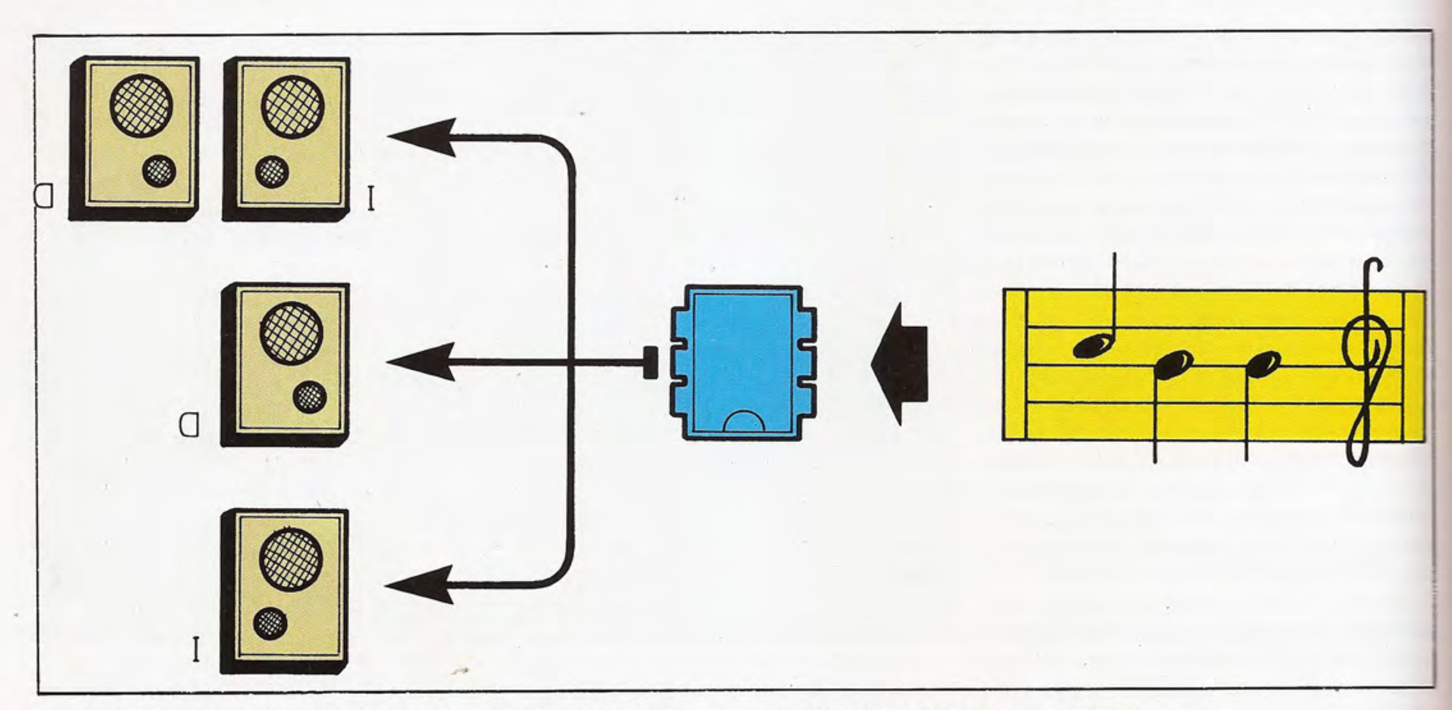
— Break ([ESC] [ESC])
— Temporizador número 3
— Temporizador número 2
— Temporizador número 1
— Temporizador número 0

Los temporizadores se controlan fundamentalmente con las órdenes AFTER («después de») y EVERY («cada») que hacen exactamente lo que sus respectivos nombres indican.

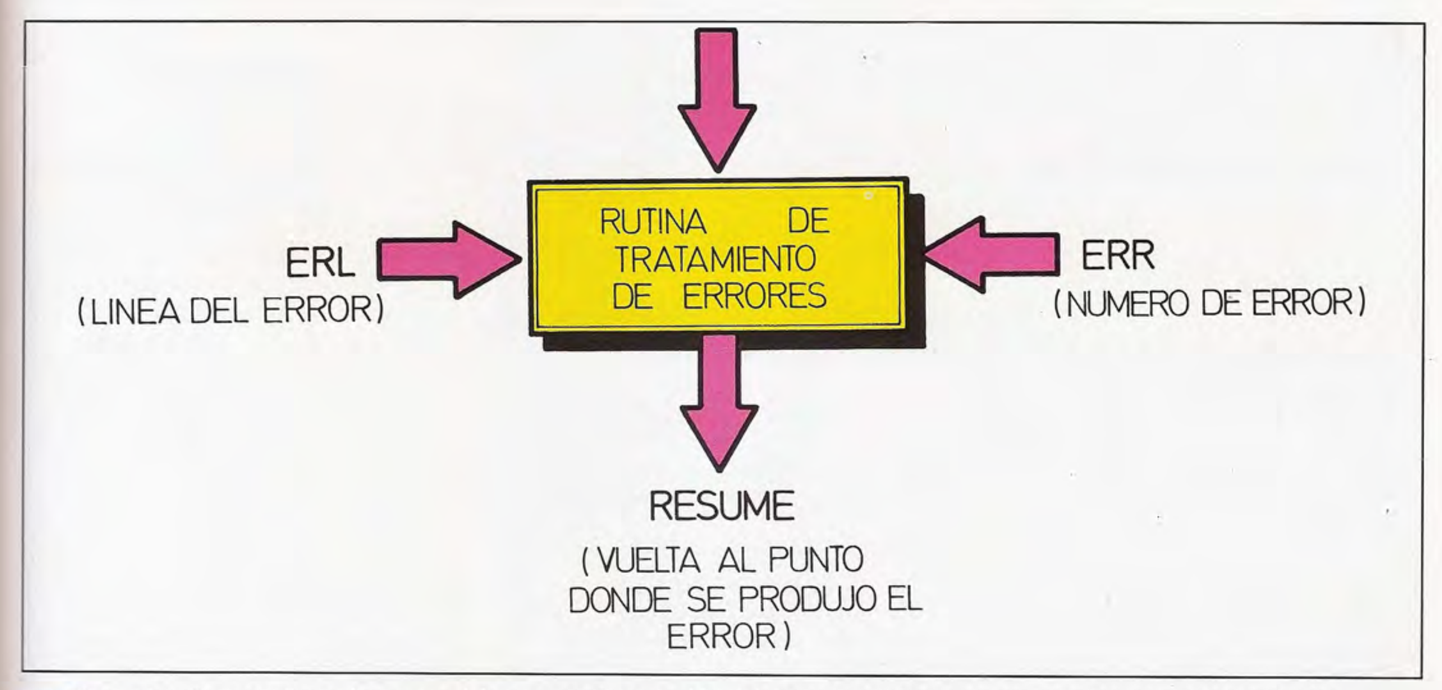
Por ejemplo, la ejecución de la instrucción

```
AFTER 500,1 GOSUB 100
```

pone a cero el temporizador número 1 y después de 500 cincuentavos de segundo (unos 10 segundos) se pasará el control a la subrutina indicada; es decir, a la que comienza en la línea 100. El efecto sería análogo al de encontrar un GOSUB 100 después de haber pasado diez segundos desde que se ejecutó el AFTER. Una vez procesada dicha subrutina




Es posible conectar a los Amstrad un amplificador estéreo que aproveche al máximo sus posibilidades acústicas. De no realizarse esta conexión, toda la discusión sobre canales carece de sentido, ya que todos los sonidos, independientemente del canal, se envían al altavoz incorporado.




*Una rutina de tratamiento de errores debe recoger información del tipo y lugar donde ocurrió el error, para poder tratar los diferentes casos de la manera oportuna. Toda rutina de este tipo acabará necesariamente con una instrucción RESUME, que devuelve el control al lugar en que se produjo el error (por supuesto, una vez que éste ha sido subsanado).*

(esto es, una vez que aparece el RETURN correspondiente) se volverá al punto del programa principal donde ocurrió la interrupción.

El funcionamiento de EVERY es análogo al anterior, con la particularidad de que la subrutina especificada será ejecutada repetidas veces. Como vemos, los temporizadores «compiten» por arrancar el control al programa principal; de ahí que sea importante establecer una jerarquía de prioridades para resolver los casos de conflicto que pudieran aparecer entre ellos.

La acción de los temporizadores puede ser inhibida sobre una sección crítica del programa a través de las instrucciones DI (Disable Interrupts) y EI (Enable Interrupts). Cuando se ejecuta una instrucción DI, los intentos de arrebatar el control por parte de los temporizadores son anulados, volviendo a la situación normal al ejecutarse una orden EI.

La propia interrupción «Break» puede se controlada por el programa en ejecución a través de una sentencia del tipo:

```
ON BREAK GOSUB 300
```

la cual bifurca hacia la subrutina indicada cuando se pulsa dos veces consecutivas la tecla ESCape. Incluso se puede inhibir por completo la posibilidad de que el operador interrumpa el programa con la instrucción ON BREAK CONT.

## El intérprete en general

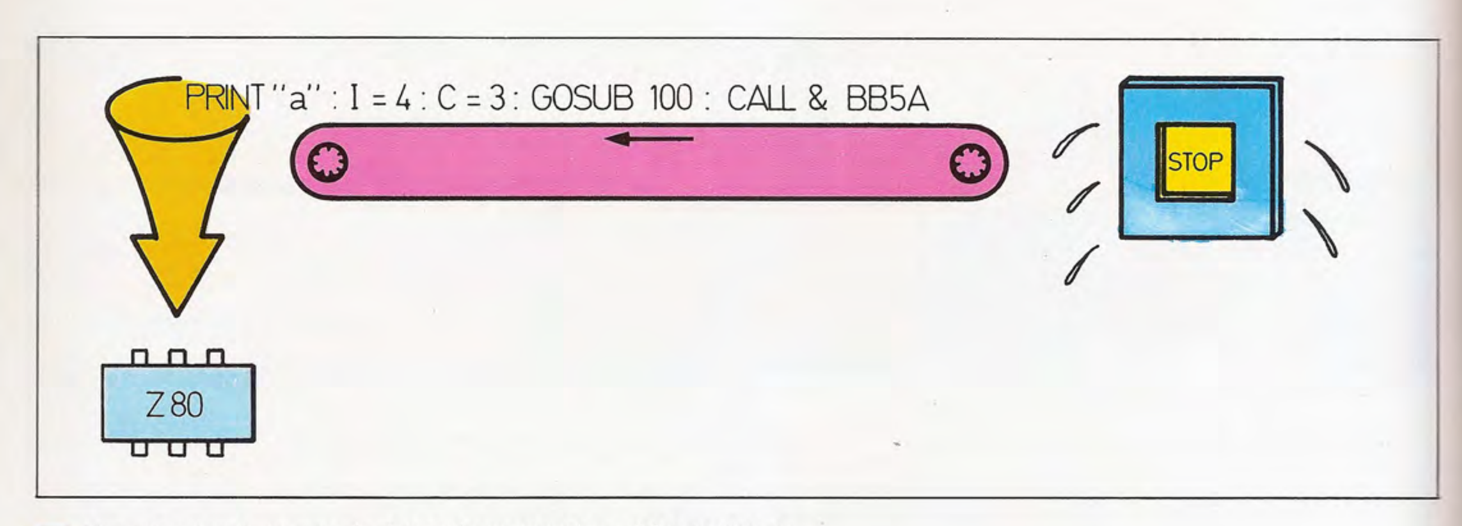
Después de este repaso, necesariamente superficial, a las características más sobresalientes de este dialecto BASIC, quedan por comentar aún una serie de pequeños detalles de interés.

Aparte de la conocida estructura FOR-NEXT para realizar bucles, existe la WHILE-WEND, que se muestra especialmente útil en la gestión de ficheros en disco/cinta. Hay que recordar que un bucle FOR-NEXT es siempre ejecutado al menos una vez, mientras que si a la entrada de un WHILE-WEND la condición es falsa, todo el bloque será saltado.

La posibilidad de tratar los errores que se puedan producir durante la ejecución del programa está ampliamente contemplada. El programador puede crear mensajes de error a la medida de la aplicación que esté desarrollando. Se pueden realizar rutinas de tratamiento de errores (ON ERROR GOTO) cuya misión consistirá en que el propio ordenador se recupere sin detener la ejecución del programa. Para ello se dispone de instrucciones específicas (ERR y ERL) que devuelven información sobre el código del error producido y la línea en que ocurrió, respectivamente; de esta forma podrá determinarse el lugar y la causa del fallo.

Si entre el gran surtido de comandos que ofrece el intérprete no se encuentra el que se necesita en un momento dado, existe la posibilidad de que el programador se lo construya a medida, a través de lo que en la jerga de Locomotive se denomina un RSX (Resident System eXtensión —Ampliación del sistema residente—).

Los RSX van precedidos por una barra vertical (|) y tienen que ser programados en código máquina. En la publicación «CPC464 Firmware Manual» se comenta la forma de programar estos comandos para poder utilizarlos desde el Basic. Por ejemplo, dada la falta de un comando en el intérprete para el dibujo de círculos, se podría pensar en la



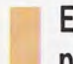

*Una interrupción provoca la detención momentánea del programa en ejecución mientras se atiende a la causa que la produjo.*

creación de un RSX cuya utilización fuera algo parecido a:

| CIRCLE,200,150,25

en el que los dos primeros parámetros sitúan el centro y el último especifica el radio. Desgraciadamente, el hecho de dotar a los Amstrad con nuevos comandos no es tarea sencilla; se necesita un buen conocimiento del lenguaje ensamblador del microprocesador Z80, así como una buena documentación sobre la forma de crearlos, la cual resulta escasa en la publicación antes citada.

Para finalizar, aparte del editor de líneas que proporcionan casi todos los intérpretes Basic para la creación de programas, existe lo que han dado en llamar un «editor de pantalla», que dista mucho de ser tal. En realidad lo que se ofrece es la posibilidad de copiar sobre la línea que está editándose el contenido total o parcial de otra línea que en ese momento aparezca en la pantalla, simplificando así la tarea de teclear una serie de líneas consecutivas con mínimas diferencias entre ellas.

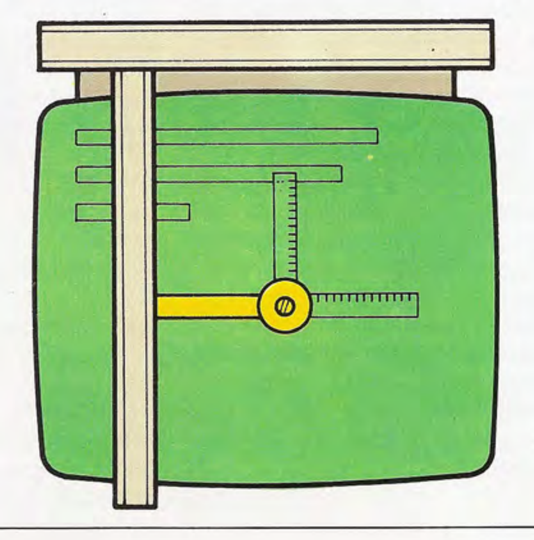


## Editores de líneas y de pantalla completa

Los editores que habitualmente acompañan a los sistemas operativos, a los programas monitores o incluso a los intérpretes de lenguajes, suelen ser de dos tipos: de líneas o de pantalla completa («full-screen»). En ambos casos, la finalidad del editor es permitir la creación y modificación de bloques de texto o de programas que deben ser sometidos a tratamiento por parte del ordenador.

Mientras que en un editor de líneas sólo es posible modificar o crear la línea que está actualmente en edición, un editor de pantalla permite moverse por cualquier punto de la misma, pudiendo realizar los cambios oportunos sin más esfuerzo que el de colocar el cursor en el punto deseado mediante las teclas pertinentes. La razón por la que la mayoría de los intérpretes de Basic incorporan un editor de líneas en vez del más cómodo de pantalla, es doble. En primer lugar, desarrollar un editor de líneas es más sencillo y ocupa menos espacio en memoria que su compañero. Además, al ser el Basic un lenguaje en el que la «línea» tiene una importancia especial —hasta el punto de que todas van numeradas, cosa que no ocurre en Pascal o C, por ejemplo—, utilizar un editor de estas características no es tan problemático como en los lenguajes citados.

ABCD 178
ORDENADOR AMSTRAD



# La máquina divertida

## Jugando con el ordenador

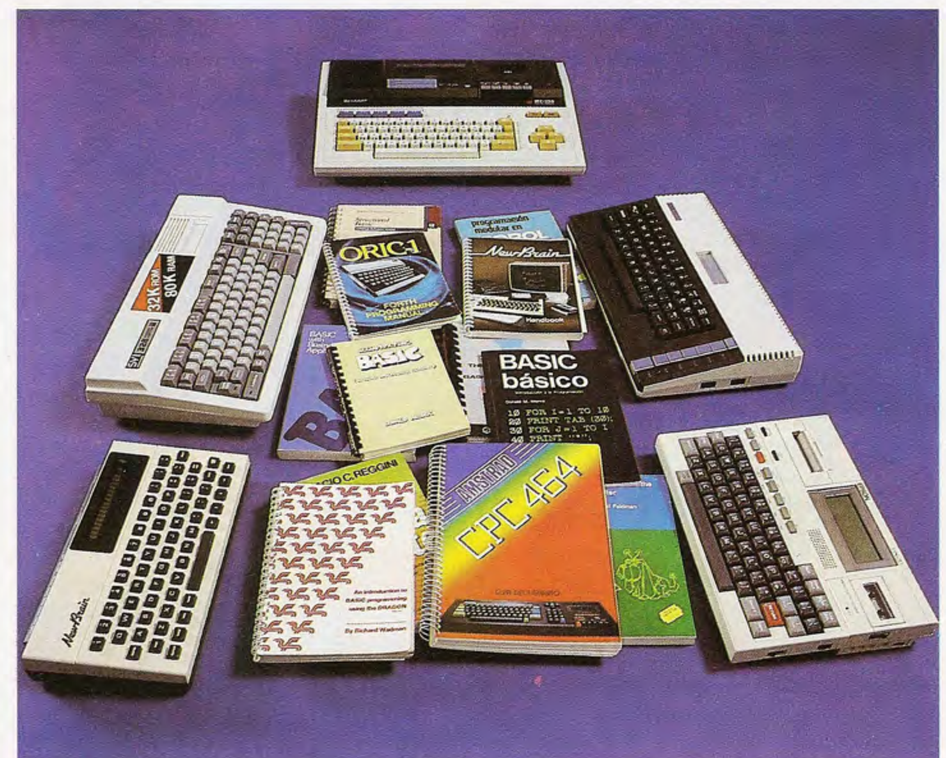


Hemos llegado casi al final de nuestro curso de BASIC. Tras el camino andado puede decirse que conocemos en profundidad los distintos comandos de este popular lenguaje. No obstante, aún falta algo: práctica. El presente capítulo, y los que quedan hasta el final del tomo, presentará diversos programas de aplicación, más o menos difíciles, pero ilustrativos de las muchas posibilidades del lengauje BASIC.

El aspecto lúdico es una de las facetas más explotadas en los ordenadores domésticos. Existe una gran cantidad de juegos comercializados para los distintos aparatos. Juegos que no sólo sirven para llenar los ratos de ocio del usuario; ahí están, por ejemplo, los juegos educativos. Con éstos, los más jóvenes, e incluso los ya adultos, ejercitan su mente de una forma amena y divertida.

La amplia proliferación de juegos para ordenador hace de éstos un moderno objeto de intercambio cultural. Sin duda, a ellos se debe en gran medida el éxito de los ordenadores domésticos.

El BASIC es un lenguaje con el que el diseño y realización de juegos resulta relativamente fácil, tal y como se observará a lo largo del presente capítulo.




*El lenguaje BASIC, presente en la mayor parte de los ordenadores personales, constituye una potente herramienta para la programación de todo tipo de aplicaciones. En éste y los posteriores capítulos desarrollaremos algunas.*

## Un juego sencillo

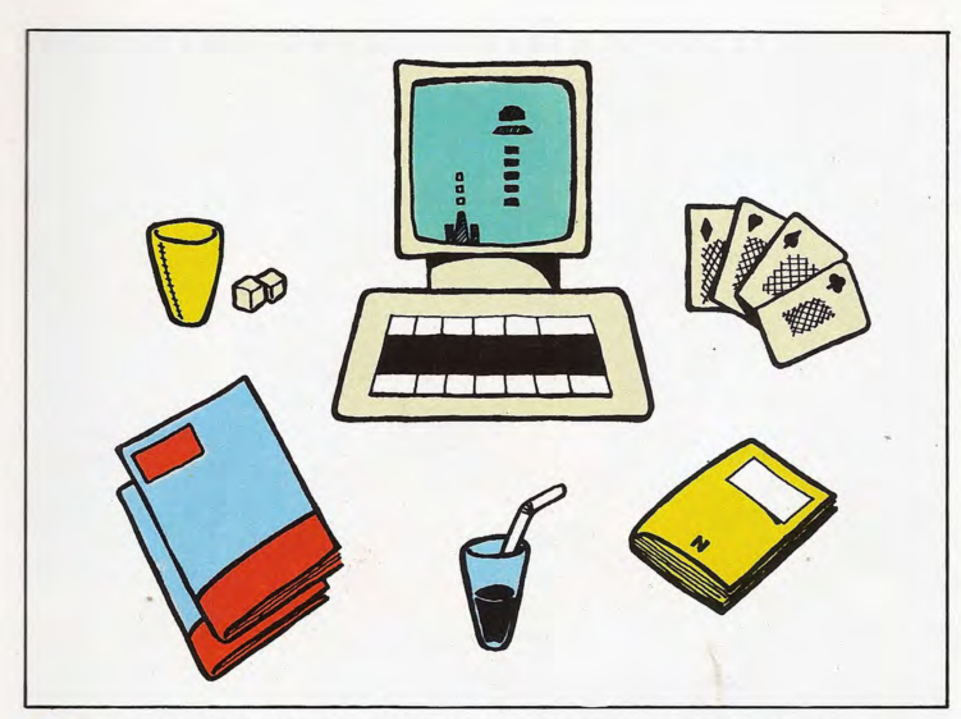
Con las herramientas que brinda el BASIC es tarea fácil construir un programa que amenice los ratos muertos del operador, por ejemplo el siguiente. El ordenador «piensa» un número, que debe ser adivinado por el jugador. Para dar una pista, el ordenador informa si el número secreto es mayor o menor que el propuesto.

La codificación en BASIC de este programa resulta de lo más simple. En primer lugar, hay que elegir el número. Este debe estar comprendido entre ciertos límites, por ejemplo 0 y 100. Para ello se hará uso de la capacidad de generación de números aleatorios del aparato, plasmada en la función RND:

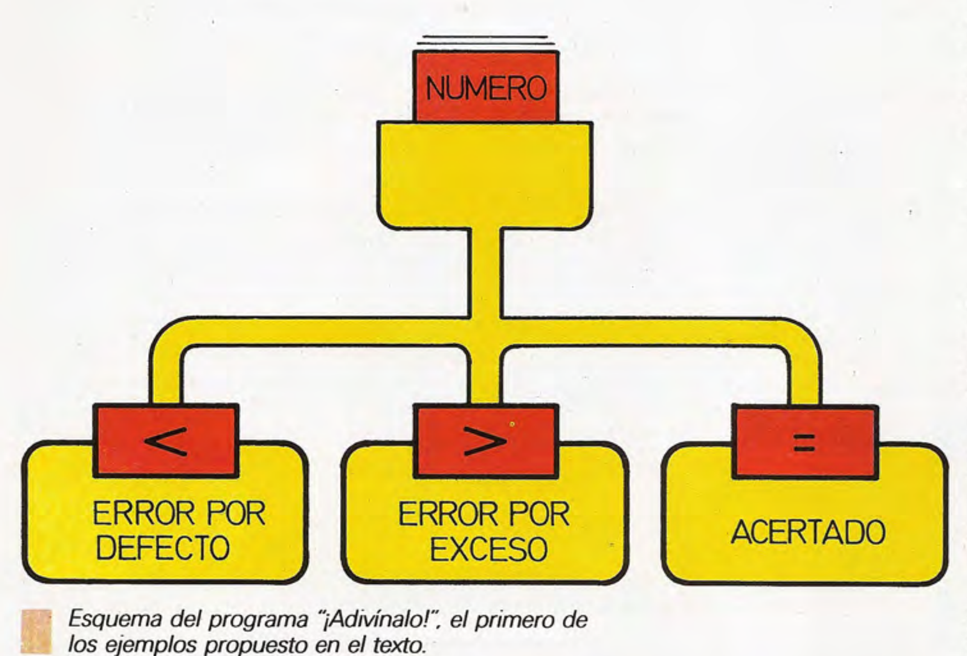
```
20 LET MINUM=INT(100*RND)
```




*El microordenador puede convertirse en un compañero de juego ideal. Sólo hace falta programarlo correctamente.*



En este primer capítulo de práctica desarrollaremos algunos programas que permiten convertir el ordenador en un eficaz compañero para los ratos de ocio.




*Esquema del programa "¡Adivínalo!", el primero de los ejemplos propuesto en el texto.*

Con esta única línea se obtiene dicho número, ya que al multiplicar RND por 100 se obtiene un valor mayor o igual que 0 y menor que 100. De éste se toma la parte entera para descartar todos los posibles valores que tuvieran parte fraccionaria. Una vez que se dispone del número a desvelar hay que brindar al usuario la posibilidad de acertarlo.

Como se tratará (muy posiblemente) de un número de un dígito no conviene utilizar INKEY$, sino que es preciso optar por el comando INPUT.

```
100 INPUT "CUAL ES"; TUNUM
```

Ahora hay que comparar el valor supuesto con el verdadero, e indicar si el número secreto es mayor o menor. En caso contrario es que ambos números será coincidentes y, en consecuencia, será preciso felicitar al jugador.

```
110 IF TUNUM>MINUM THEN PRINT "NOOO!! ES ME
    NOR"
120 IF TUNUM<MINUM THEN PRINT "NOOO!! ES MA
    YOR"
130 IF TUNUM=MINUM THEN PRINT "MUY BIEN,ERA EL
    ";MINUM
```

Con estas líneas se consiguen distintos mensajes dependiendo del número aportado. Si no se acierta, al menos se obtiene una pista. Aunque de poco sirve esa pista si no se pueden seguir probando números. Si el intento es infructuoso, habrá que repetirlo. Es evidente pues que hace falta un GOTO 100. Para situarlo debidamente es necesario hacer una pequeña reestructuración:

```
130 IF TUNUM=MINUM THEN GOTO 200
140 GOTO 100
```

Ahora, el número correcto desviará la ejecución hacia la línea 200. Si el número es mayor o menor (distinto) del pensado por la máquina, se efectuará el salto a la línea 100.

A partir de la línea 200 hay que escribir la felicitación y dar por concluido el juego.

```
200 PRINT "MUY BIEN, ERA EL ";MINUM
```

Aunque un detalle de buen gusto consiste en permitir al usuario que repita el juego (con otro número, por supuesto). Al efecto se introducirán las líneas oportunas que permitan al operador elegir entre volver a jugar o abandonar el programa. Estas son:

```
210 PRINT"JUGAMOS OTRA VEZ (S/N)?"
220 LET A$=INKEY$
```

```
230 IF A$="S" THEN GOTO 20
240 IF A$<>"N" THEN GOTO 220
250 PRINT"HASTA LA PROXIMA!"
```

Las líneas 210 y 220 se encargan de recoger la petición del jugador. Si éste responde con "S" (Sí) se reinicia el programa en la línea 20. De no ser ésta su respuesta, el programa salta a la línea 240. En ella se bifurca a la línea 220 si no conteso con una "N". Por lo tanto, si no se responde con "S" o "N", se recogerá un nuevo carácter del teclado. Cuando la respuesta es una pulsación de la tecla N, se continúa en la línea 250. En tal caso, el ordenador se despide dando por concluida la ejecución del programa.

Por último, sería interesante indicar al principio del juego el rango de números a considerar como posibles. Para ello basta con incluir la siguiente línea:

```
30 PRINT"HE PENSADO UN NUMERO DEL CERO AL CIEN"
```

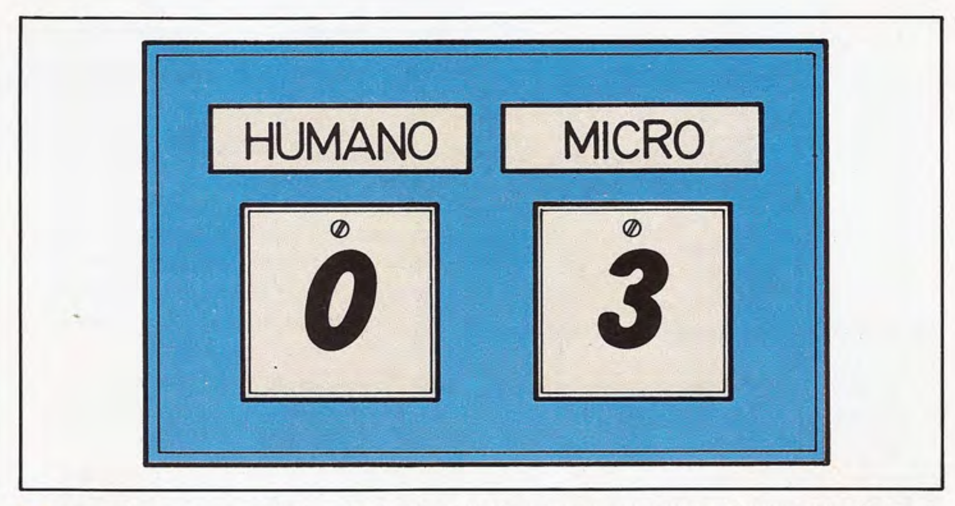
El programa completo es el que aparece en el cuadro adjunto bajo el título "ADIVINALO!".

Una posible partida seguirá la evolución siguiente:

```
RUN
HE PENSADO UN NUMERO DEL CERO AL CIEN
CUAL ES? 52<CR>
NOOO!! ES MENOR
CUAL ES? 23<CR>
NOOO!! ES MAYOR
CUAL ES? 45<CR>
MUY BIEN,ERA EL 45
JUGAMOS OTRA VEZ (S/N)? N
HASTA LA PROXIMA!
```




*¿Se siente capaz de adivinar el número «pensado» por su ordenador?*




*No piense que puede hacer ningún tipo de trampas, ni variar el tanteo a su favor. El ordenador es implacable en todos los aspectos.*

## Una partida a los chinos

Adoptando algunas de las ideas plasmadas en el ejemplo anterior, es posible redactar un nuevo programa con el que el usuario se puede jugar el aperitivo. Se trata, esta vez, de enseñar al ordenador a jugar al popular deporte de «los chinos». Ello se reduce a adivinar un número del 0 al 3 y sumarlo al previamente pensado. El problema se puede desglosar en varias partes:

— Elegir un número (del 0 al 3)

— Pedir vaticinio del jugador (del 0 al 6)

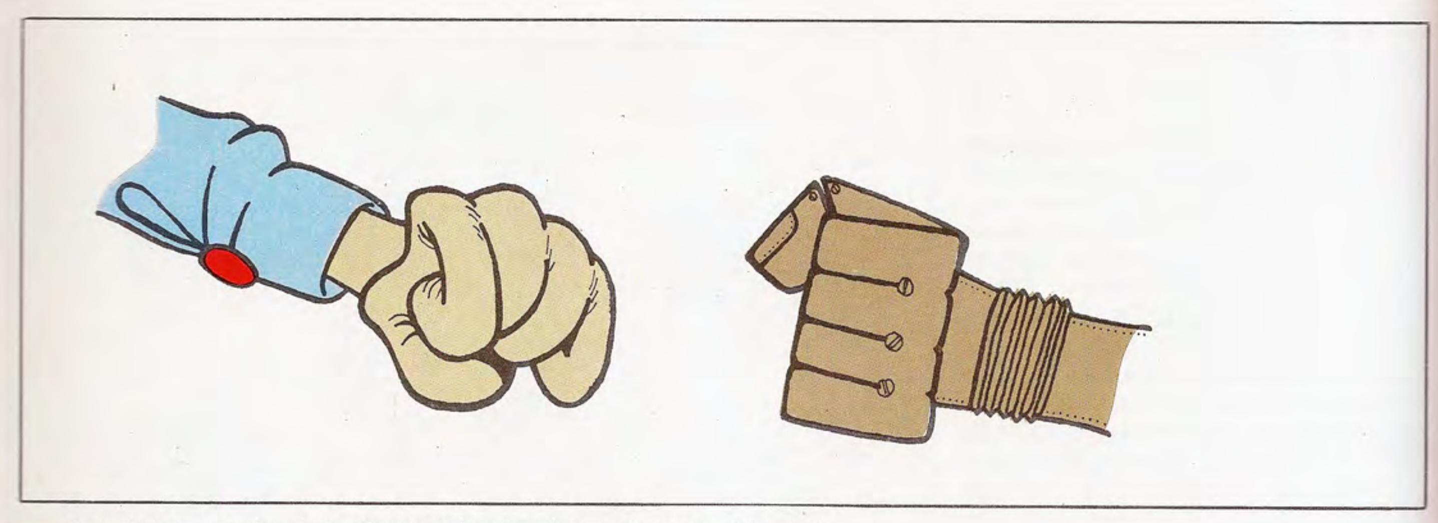
— Calcular el rango de valores posibles

— Elegir un número dentro de ese rango.

Los dos primeros pasos son similares a los del ejemplo anterior. La única variación está en los límites del valor elegido:

```
20 LET MIAS=INT(4*RND)
30 INPUT "CUANTAS" ;T
```

El margen de posibles valores será, en principio, desde MIAS hasta MIAS+3. Para elegir un valor aleatorio dentro de



*Otro de los ejemplos de la eficacia del ordenador en el terreno del ocio lo proporciona el programa capaz de instruir a la máquina para que juegue a los chinos.*

estos límites basta con la siguiente línea:

```
50 LET Y=MIAS+INT(4*RND)
```

Este cálculo debe repetirse hasta encontrar un número distinto del proporcionado por el adversario, lo que se consigue introduciendo un comando IF:

```
60 IF Y=T THEN GOTO 50
```

Si el número Y elegido coincide con T (el pronóstico del oponente) se vuelve a elegir un nuevo Y. Sólo se continuará en el caso de que ambos datos difieran.

El programa completo necesita un par de líneas más para hacer público su pronóstico y mostrar su jugada (líneas 70 y 80):

```
10 REM CHINOS 1/SOFT
20 LET MIAS=INT(4*RND)
30 INPUT "CUANTAS";T
50 LET Y=MIAS+INT(4*RND)
60 IF Y=T THEN GOTO 50
70 PRINT "DIGO QUE HAY ";Y
80 PRINT "LLEVO ";MIAS
█
```

Desde luego, se puede sacar alguna información adicional del dato proporcionado por el contrincante. Si pide 0 es que no lleva ninguna, si pide 1 es que lleva 0 ó 1, etc. Esto se puede abreviar diciendo que si el contrario pide N es que sólo puede llevar un número comprendido entre 0 y N. Axioma éste sólo válido para N menor que 4.

Si por el contrario estima que el número correcto es 4, es porque él lleva al menos una. Con esta lógica se observa que, dependiendo del dato adquirido, se puede limitar el rango de valores.

Así pues, los límites de los posibles valores serán:

```
DATO      0
0         01
1         012
2         0123
3          123
4           23
5            3
6
```

— *Mínimo:* N—3 ó 0
— *Máximo:* N ó 3

Lo que convertido en líneas de programa, se traduce en:

```
100 LET MIN=T—3
110 IF MIN<0 THEN MIN=0
120 LET MAX=T
130 IF MAX>3 THEN MAX=3
```

Las líneas 100 y 120 calculan los valores máximo y mínimo según las fórmulas comentadas más arriba. Las otras dos líneas, 110 y 130, impiden que dichos valores se sitúen por debajo de 0 (el mínimo) o por encima de 3 (el máximo).

Una vez obtenido el rango apropiado, es preciso elegir uno de los números admisibles. Al respecto se utilizará la fórmula comentada en el capítulo dedicado a la función RND:

```
INT(RND*(MAX—MIN1))+MIN
```

En ella, todos los valores son conocidos. Lo que interesa es adivinar el número total de monedas, por lo que, al número hallado, ha de sumársele aquel que juega la máquina. Todo ello se almacena en la variable Y para su posterior manipulación.

```
200 LET Y=MIAS+INT(RND*(MAX—MIN+1))+MIN
```

Claro que el pronóstico así evaluado puede coincidir con el del contrincante. Si eso ocurre habrá que buscar otro valor, recurriendo al mismo método utilizado en el programa anterior:

```
210 IF Y=T THEN GOTO 200
```

Esta simple línea hace que se repita el cálculo hasta que el número resultante sea distinto del proporcionado por el jugador. No obstante, esta actuación plantea un inconveniente: si el rango de valores posibles se limita a un solo número, y éste coincide con el elegido por el adversario, la ejecución entrará en un bucle cerrado.

En efecto, el ordenador calculará siempre el mismo valor Y que coincide con T, y saltará de nuevo a la línea 200.

Este tipo de bucles que no tienen salida (al igual que los del tipo 100 GOTO 100) se denominan «bucles infinitos». El programador debe huir de estos bucles, ya que introducen al programa en un «callejón sin salida».

denador no le queda más remedio que rendirse ante la evidencia de su derrota.

```
220 IF Y=T THEN PRINT "ACERTASTE"
```

La otra posibilidad es que seleccione un valor distinto al del contrincante (T e Y son distintos). Ello indica que todo va bien y el ordenador puede emitir su pronóstico y mostrar su jugada:

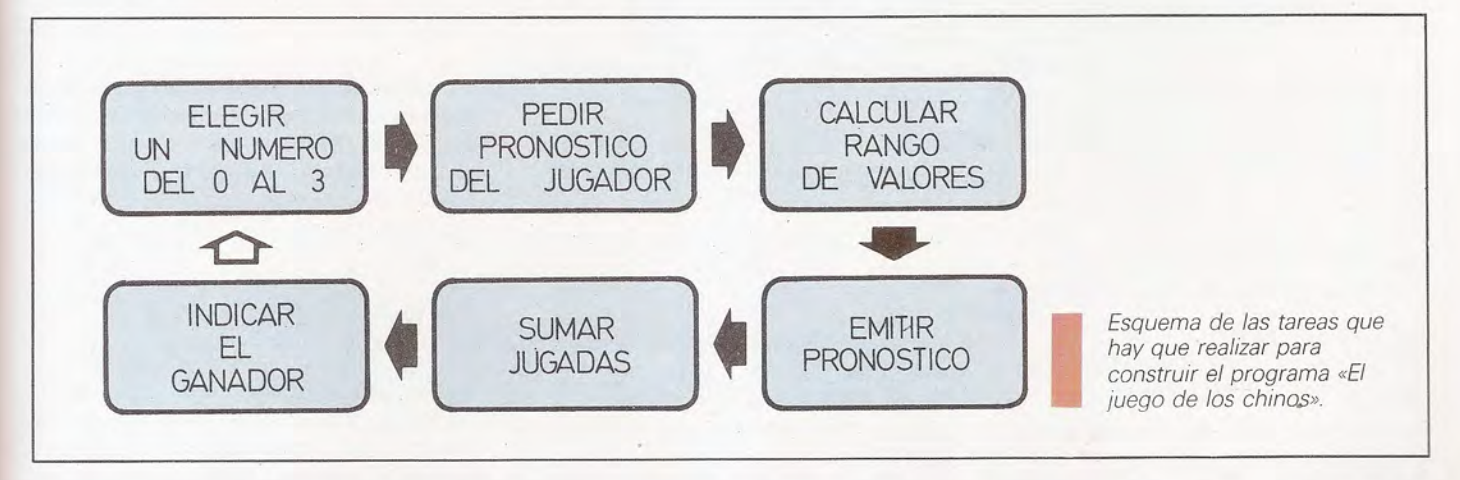
```
230 IF Y<>T THEN PRINT "YO DIGO "; Y
300 PRINT "LLEVO ";MIAS
```

Si el usuario quiere ahorrarse el trabajo de contar las fichas de uno y otro participante e ir llevando el tanteo acumulado, ha de completar el programa con algunas líneas más. En primer lugar, el ordenador debe conocer el número de fichas jugado por el oponente, para hallar la suma total. La recogida de este dato se realizará por medio de un comando INPUT.

```
290 INPUT "CUANTAS LLEVAS";TUYAS
```

Como se observa, esta línea se ha colocado inmediatamente antes de que el aparato muestre su jugada (línea 300). Así se evitan cambios de última hora por parte del operador humano.

Una vez conocido este dato, hay que proceder a su suma y posterior comparación con los pronósticos de ambos jugadores.




*Esquema de las tareas que hay que realizar para construir el programa «El juego de los chinos».*

Así pues, sólo debe recalcularse el pronóstico cuando el rango incluya al menos 2 valores posibles. O lo que es lo mismo, cuando MIN y MAX tomen valores distintos. La anterior línea 200 se sustituiría, en consecuencia, por la que sigue:

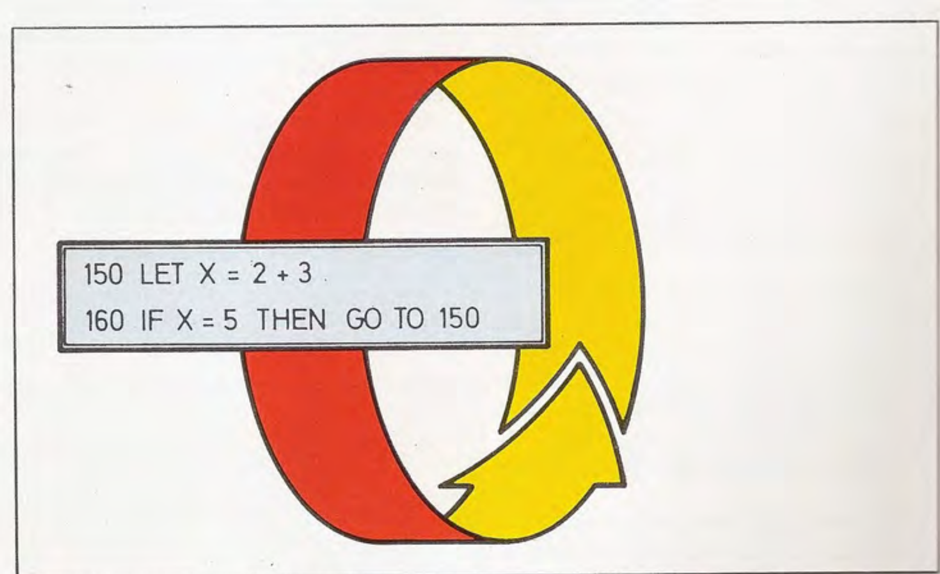
```
210 IF Y=T AND MIN<>MAX THEN GOTO 200
```

Ahora la ejecución puede continuar de dos posibles formas:

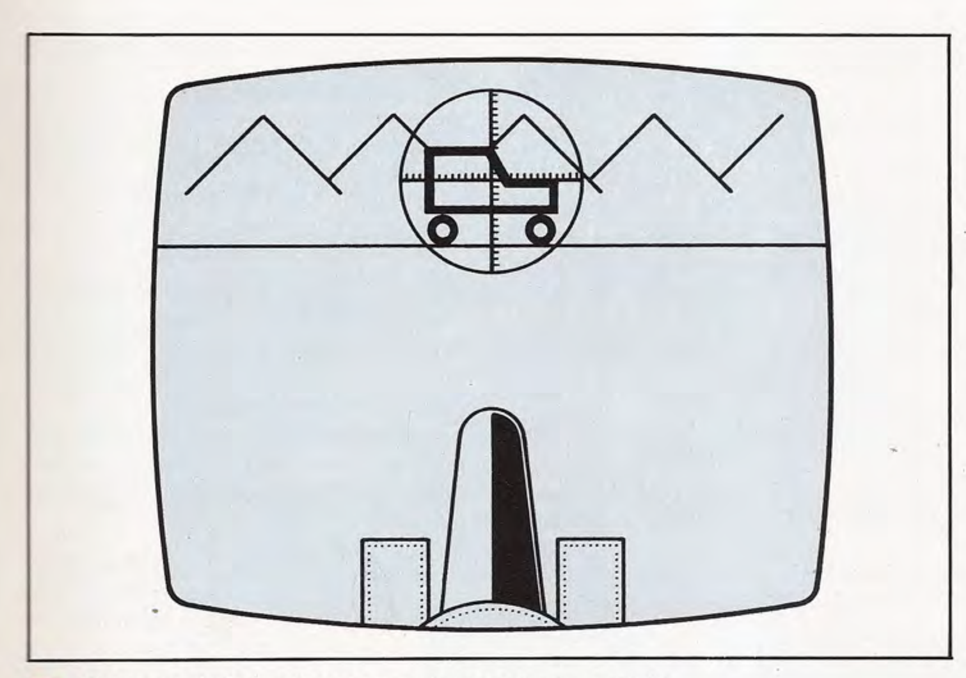
— Al calcular un número distinto del ofrecido por el oponente.

— Cuando el contrario ha elegido el único valor posible.

En el segundo caso no cabe duda de que el jugador humano ha acertado. Dicha situación se dará cuando, al pasar a la siguiente línea, los valores de T e Y sean idénticos. En ese momento, al or-




*Los bucles infinitos son estructuras cíclicas que encierran el flujo del programa, impidiendo que evolucione la secuencia de proceso. El programador debe evitar su formación accidental.*



*Escenario de la batalla programada como tercer ejemplo de juego con el ordenador.*

```
310 LET S=TUYAS+MIAS
320 IF S=Y THEN PRINT "GANO YO"
330 IF S=T THEN PRINT "GANAS TU"
```

En las líneas 320 y 330 es donde se compara el resultado final con los números propuestos por los jugadores. Si alguno de ellos acierta, se informa quién es el ganador.

También es posible llevar la cuenta de los aciertos de cada contendiente. Para ello basta con incrementar un contador particular cada vez que tenga lugar ese hecho. El lugar apropiado para introducir las órdenes pertinentes es el argumento de THEN de las líneas 320 y 330. Para poder incluir en ellos tanto la impresión del mensaje como la actualización del contador, se hará uso de la posibilidad de encadenar varias órdenes en la misma línea. Esto, como ya se vio, se realiza separándolas con el símbolo dos puntos (:).

Así pues, en las dos nuevas líneas las variables MI y HU llevan la cuenta de las partidas ganadas por el «micro» y el «humano», respectivamente.

```
320 IF S=Y THEN PRINT "GANO YO": LET MI=MI+1
330 IF S=T THEN PRINT "GANAS TU":LET HU=HU+1
400 PRINT "HUMANO:";HU;"MICRO:";MI
```

Se ha incluido la línea 400 que muestra la puntuación alcanzada por los jugadores hasta ese momento. Para evitar que ninguno de los participantes empiece con ventaja, es necesario inicializar a cero ambos contadores al principio de la partida:

```
10 LET HU=0:LET MI=0
```

Por último, al final del programa hay que incluir una instrucción que permita volver a jugar. El salto se realiza a la línea 20, en la que empezará una nueva partida.

```
1000 GOTO 20
```

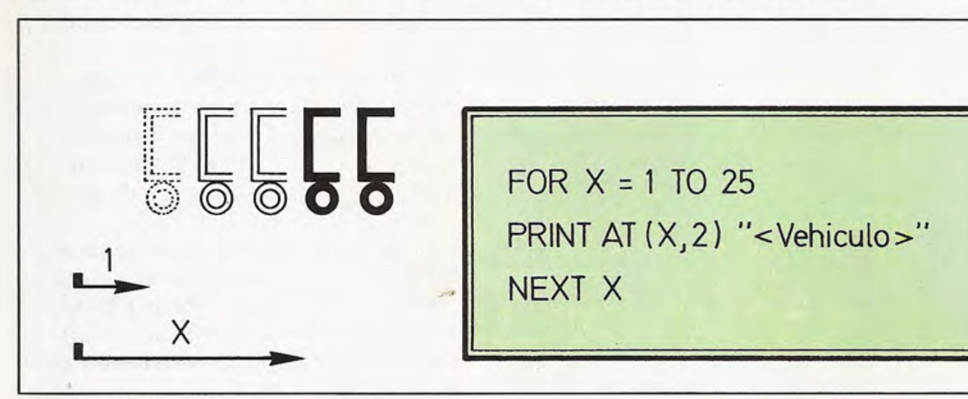
Queda al gusto del lector la inclusión de una rutina de espera entre las líneas 400 y 1000; rutina que otorgue el tiempo suficiente para elegir la siguiente jugada. El programa completo (CHINOS) aparece listado en el cuadro adjunto.

## Un juego de habilidad

El siguiente programa consiste en un juego de habilidad. El jugador ha de disparar a un blanco móvil. Dicho blanco es un transporte enemigo que debe ser destruido a toda costa.

El problema se va a afrontar con un criterio de modularidad, diseñando cada parte por separado, para al final unirlas todas.

El vehículo se desplazará a través del borde superior de la pantalla. Ello significa que habrá que dibujar el blanco sucesivamente en las distintas posicio-




*El movimiento del vehículo enemigo se simula emplazando a éste, sucesivamente, a mayor distancia del borde izquierdo de la pantalla.*

nes. Las líneas que dibujan el vehículo son las siguientes:

```
110 PRINT AT(X,Y) "LLL"
120 PRINT AT(X,Y) "O O"
```

Los parámetros Y de la función AT adoptarán los valores 1 y 2, respectivamente, para que el blanco se mantenga en las dos líneas superiores. El parámetro X variará desde 1 hasta el valor que lleve al vehículo al extremo de la pantalla. La variación se consigue encerrando ambas instrucciones dentro de un bucle FOR/NEXT.

```
100 FOR X=1 TO 25
110 PRINT AT(X,1) "LLL"
120 PRINT AT(X,2) "O O"
130 NEXT X
```

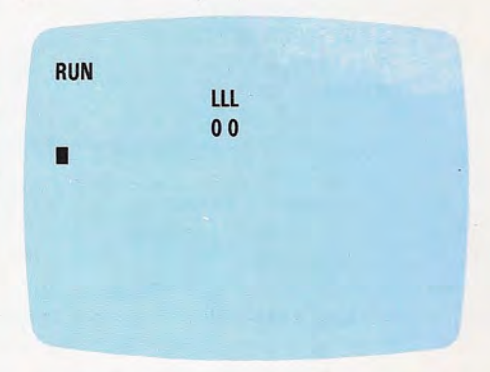
Esta zona del programa hará que se desplace el blanco. Pero, al no borrarse la posición anterior persistirá en la pantalla el rastro del recorrido:

```
RUN
LLLLLLLLLL LL
0000000000 0 0
■
```

Esto se soluciona introduciendo, sencillamente, un espacio en blanco al principio de cada una de las cadenas que representan al vehículo.

```
100 FOR X=1 TO 25
110 PRINT AT(X,1) " LLL"
120 PRINT AT(X,2) " O O"
130 NEXT X
```

De esta forma, los espacios borrarán la imagen de las posiciones precedentes.




Con el enemigo ya funcionando, lo que hay que hacer ahora es dibujar nuestro cañón. Se tomará como representación del mismo un carácter cuyo aspecto recuerde la forma del mismo. La letra L puede servir para ese propósito. Esta se colocará en un punto central de la zona inferior de la pantalla. Por ejemplo, en la columna 15 de la vigésima fila. Dichos valores, al igual que el 25 del bucle FOR anterior, pueden ser modificados al gusto del usuario. La línea que sitúa el cañón puede ser la siguiente:

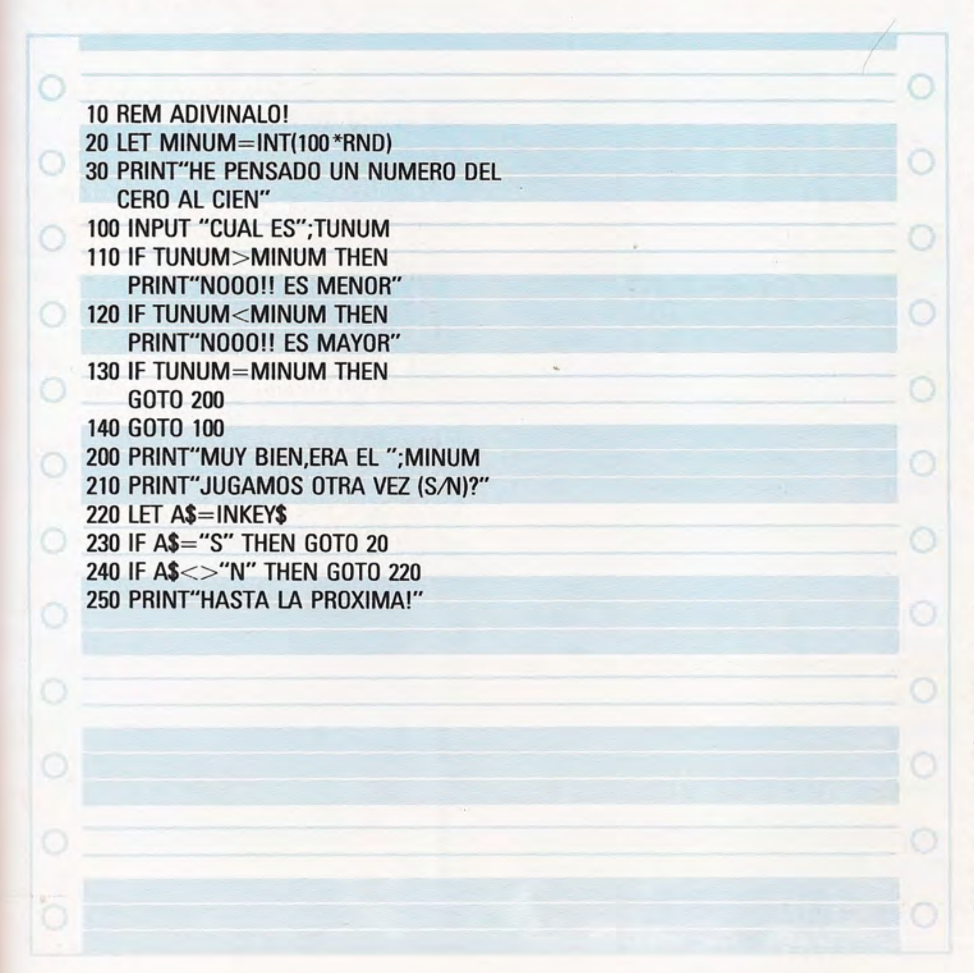
```
50 PRINT AT(17,20)"I"
```

Ahora viene la rutina de disparo. Esta se utilizará a modo de subrutina, por lo que es conveniente separarla del resto del programa.

```
1000 FOR I=19 TO 2 STEP -1
1005 FOR J=0 TO 2
1010 PRINT AT(17,Z)"!"
1020 PRINT AT(17,Z)" "
1025 NEXT J
1030 NEXT I
```

Esta rutina traza la trayectoria del proyectil empleando el carácter !. Su desplazamiento tiene lugar a lo largo de la columna 17 de la pantalla, columna que corresponde a la posición del cañón. El bucle en J hará que el proyectil parpadee en cada posición antes de desaparecer.

La rutina de disparo únicamente se ha de ejecutar cuando se pulse una tecla. Para ello, es preciso incluir algunas líneas que reconozcan la pulsación. Líneas que deben estar localizadas en la zona del programa que mueve el blan-



```
10 REM ADIVINALO!
20 LET MINUM=INT(100*RND)
30 PRINT"HE PENSADO UN NUMERO DEL
   CERO AL CIEN"
100 INPUT "CUAL ES";TUNUM
110 IF TUNUM>MINUM THEN
    PRINT"NOOO!! ES MENOR"
120 IF TUNUM<MINUM THEN
    PRINT"NOOO!! ES MAYOR"
130 IF TUNUM=MINUM THEN
    GOTO 200
140 GOTO 100
200 PRINT"MUY BIEN,ERA EL ";MINUM
210 PRINT"JUGAMOS OTRA VEZ (S/N)?"
220 LET A$=INKEY$
230 IF A$="S" THEN GOTO 20
240 IF A$<>"N" THEN GOTO 220
250 PRINT"HASTA LA PROXIMA!"
```

```
5 REM CHINOS
10 LET HU=0:LET MI=0
20 LET MIAS=INT(4*RND)
30 INPUT "CUANTAS";T
100 LET MIN=T-3
110 IF MIN<0 THEN MIN=0
120 LET MAX=T
130 IF MAX>3 THEN MAX=3
200 LET Y=MIAS+INT(RND*(MAX-MIN
    +1))+MIN
210 IF Y=T AND MIN<>MAX THEN
    GOTO 200
220 IF Y=T THEN PRINT "ACERTASTE"
230 IF Y<>T THEN PRINT "YO DIGO ";Y
290 INPUT "CUANTAS LLEVAS";TUYAS
300 PRINT "LEVO";MIAS
310 LET S=TUYAS+MIAS
320 IF S=Y THEN PRINT "GANO YO":LET
    MI=MI+1
330 IF S=T THEN PRINT "GANAS TU":LET
    HU=HU+1
400 PRINT "HUMANO:";HU;"MICRO:";MI
1000 GOTO 20
```

co; en caso contrario, el disparo sólo podría efectuarse cuando el enemigo ya ha rebasado la posición del cañón.

```
122 LET T$=INKEY$
124 IF T$<> THEN GOSUB 1000
```

La rutina de disparo no está completa aún. Es necesario detectar si se ha producido una colisión entre el proyectil y el blanco. Tal operación se realiza al final de la rutina de disparo. Lo único que hay que hacer es cerciorarse de que el objetivo se encuentra (o no se encuentra) en la vertical del cañón. Este hecho se produce cuando la variable X adquiere los valores 16, 15 ó 14.

Así pues, se plantea una condición que de cumplirse indicará que se ha alcanzado el objetivo:

```
1100 IF X>13 AND X<17 THEN GOTO 2000
1110 RETURN
```

El efecto de ambas líneas desvía la ejecución hacia la línea 2000 si se produce la colisión, u ordena el retorno al programa principal en el caso contrario. Su actuación permite el tratamiento aparte (línea 2000) de la colisión. Dicho tratamiento consistirá en visualizar la explosión y reiniciar el programa; desde luego, también se puede actualizar un contador que indique el número de blancos acertados.

```
2000 FOR J=0 TO 50
2010 PRINT AT(17,1)"*"
2020 PRINT AT(17,1)" "
2030 NEXT J
2040 LET PUNTOS=PUNTOS+1
2050 X=25
2060 RETURN
```

En estas últimas líneas se ha incluido un bucle (líneas de la 2000 a la 2030) que imprime y borra, alternativamente, un asterisco en el punto de impacto del proyectil. Ello produce un efecto de explosión. El bucle se repite el número de veces suficiente como para que se aprecien los daños producidos.

La línea 2040 es la encargada de incrementar en una unidad el controlador de aciertos. Como se puede observar, la actualización de PUNTOS sólo se efectúa en el caso de alcanzar el objetivo.

La línea 2050 del listado pone el valor de X a 25. Con ello se consigue dar por terminado el bucle comprendido entre las líneas 100 y 130, al retornar de la subrutina. De esta forma, el carro enemigo alcanzado no continúa su marcha.

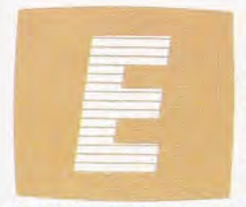
Por último, la línea 2060 proporciona el punto de retorno de la subrutina, en el supuesto de que no se haya producido el regreso en la 1110.

El programa completo (BATALLA) exhibirá el aspecto que muestra el cuadro adjunto. La línea 30 limpia la pantalla para evitar que se superponga el escenario del juego a cualquier texto anterior. La siguiente línea (40) muestra los aciertos en la zona inferior izquierda de la pantalla. La variable PUNTOS es inicializada en la línea 20. Por fin, la línea 150 reinicia la ejecución cuando no se destruye el objetivo.

```
10 REM BATALLA
20 LET PUNTOS=0
30 CLS
40 PRINT AT(1,20)"PUNTOS ";PUNTOS
50 PRINT AT(17,20)"I"
100 FOR X=1 TO 25
110 PRINT AT(X,1) " LLL"
120 PRINT AT(X,2) " 0 0"
122 LET T$=INKEY$
124 IF T$<>" "THEN GOSUB 1000
130 NEXT X
150 GOTO 30
1000 FOR I=19 TO 3 STEP -1
1005 FOR J=0 TO 2
1010 PRINT AT(17,I)"!"
1020 PRINT AT(17,I)" "
1025 NEXT J
1030 NEXT I
1100 IF X>13 AND X<17 THEN GOTO 2000
1110 RETURN
2000 FOR J=0 TO 50
2010 PRINT AT(17,1)"*"
2020 PRINT AT(17,1)" "
2030 NEXT J
2040 LET PUNTOS=PUNTOS+1
2050 X=25
2060 RETURN
```

# Juegos «a medida»

## El campo minado



n este nuevo capítulo, dedicado a la práctica del lenguaje BASIC, se acometerá la confección, paso a paso, de un programa adecuado para ejecutar el tradicional juego del «campo minado».

Como apunta el nombre, el objetivo del juego es cruzar con éxito el peligroso trayecto, sorteando las minas ocultas que pueden hacer saltar a nuestro personaje por los aires.

Como instrumento de ayuda, el jugador dispone de un detector de minas que le informa, mediante un pitido, de la inmediata proximidad de un artefacto explosivo. Para cruzar el campo minado, el personaje puede desplazarse en cuatro sentidos: arriba, abajo, derecha e izquierda.

El proceso a seguir para el desarrollo del programa se expondrá con toda suerte de detalles en los próximos párrafos. El diagrama de flujo adjunto resume la estructura general del programa a confeccionar.




*El objetivo del presente capítulo, dedicado a la práctica del BASIC, es confeccionar un programa que permita ejecutar el tradicional juego del «campo minado».*

*El movimiento del personaje a través del campo minado bajo el control de las teclas A, W, D, y X.*

### Preparación de la pantalla

Para empezar es conveniente preparar la pantalla de forma adecuada. Entre las operaciones que conducirán a tal preparación se encuentra el borrado de la pantalla y la selección del modo gráfico más idóneo.

```
10 REM JUEGO DEL CAMPO MINADO
20 CLS:SCREEN 2
```

La línea 20 es la encargada de borrar la pantalla por medio del comando CLS. Evidentemente, en los ordenadores que no posean esta instrucción será necesario sustituirla por su equivalente. Al efecto, es conveniente recordar la existencia de los códigos de control, alguno de los cuales permitirá realizar esta misma acción.

Los códigos de control son propios de cada aparato; en consecuencia, no es posible precisar en estas páginas el código específico asociado a cada ordenador. No obstante, sí cabe mencionar que uno de los más corrientes es el 26, de manera que ejecutando la siguiente instrucción:

```
PRINT CHR$(26)
```

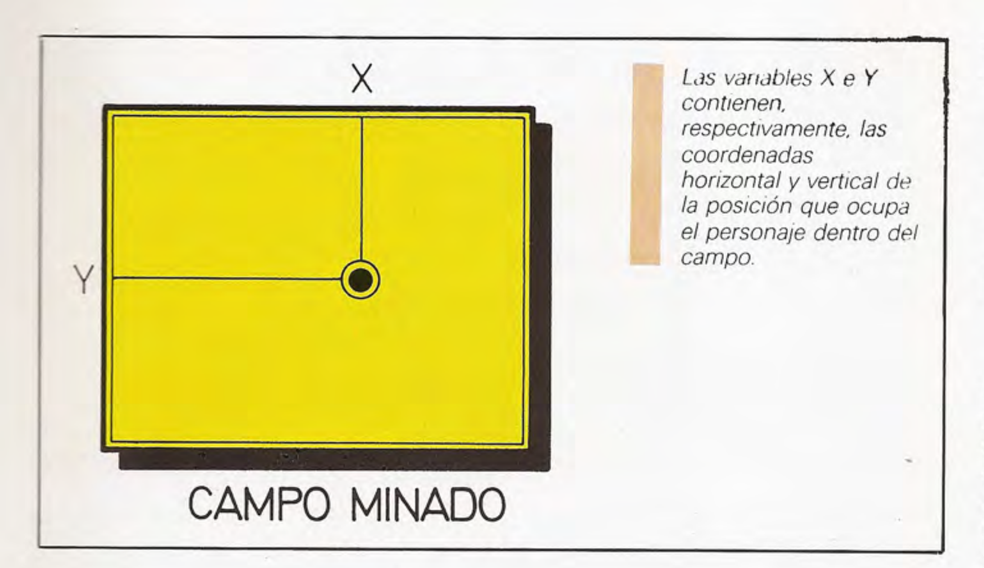
se obtendrá como resultado el borrado de la pantalla.

También es posible que no se emplee un único código para el borrado de la presentación visual, sino que el equipo exija la puesta en práctica de una combinación de códigos; por ejemplo:

```
PRINT CHR$(27)+"E"
```

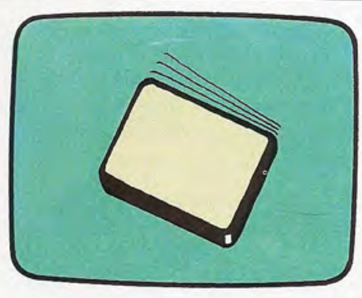
En la propia línea 20 del programa también es posible seleccionar el modo gráfico adecuado para que en él se desarrolle el juego. En este caso, sí es totalmente obligado remitir al lector al manual de su ordenador, ya que las diferencias son drásticas de uno a otro equipo.

Si el equipo utilizado es un ordenador personal del tipo MSX, IBM-PC o compatible, será necesario en este punto eliminar de la pantalla la presentación que ocupa su zona inferior, presentación




*Las variables X e Y contienen, respectivamente, las coordenadas horizontal y vertical de la posición que ocupa el personaje dentro del campo.*

*Para facilitar el movimiento a través del campo minado (o para entorpecerlo, poniendo nervioso al jugador), se dispone de un detector que avisa de la proximidad de alguna mina*






*La primera tarea que hay que programar consiste en el borrado de la pantalla, bien mediante el comando CLS, bien mediante alguna orden equivalente propia del dialecto BASIC que estemos utilizando.*

que recuerda los comandos disponibles en las teclas de función. Ello se logra, sencillamente, ejecutando la siguiente instrucción:

```
40 KEY OFF
```

Claro está que si esta franja de comandos no aparece en la pantalla de su ordenador, resultará inútil la inclusión de este comando.

## Minado del campo

Para saber en qué puntos del campo se encuentran las minas, recurrimos al artificio de crear una matriz con tantos elementos como posibles casillas pueda recorrer nuestro personaje. Acto seguido, es preciso que el programa coloque las minas en el terreno en el que se va a desarrollar el juego, adoptando un criterio aleatorio:

```
 50 DIM A(31,21)
 60 REM COLOCACION DE LAS MINAS
 70 FOR I=1 TO 40
 80 X2=1+INT(RND*30)
 90 Y2=1+INT(RND*20)
100 X2=1 AND Y2=1 THEN GOTO 110 ELSE A(X2,Y2)=1
110 NEXT I
```

Esta rutina emplaza las minas de forma aleatoria a lo largo del escenario de juego. El número de minas depositadas se eleva a 40; sin embargo, difícilmente se podrán encontrar todas estas minas en el campo a cruzar, puesto que no se ha previsto ningún control para evitar que coincidan dos minas en el mismo lugar. Este hecho hará que aumente la aleatoriedad del juego.

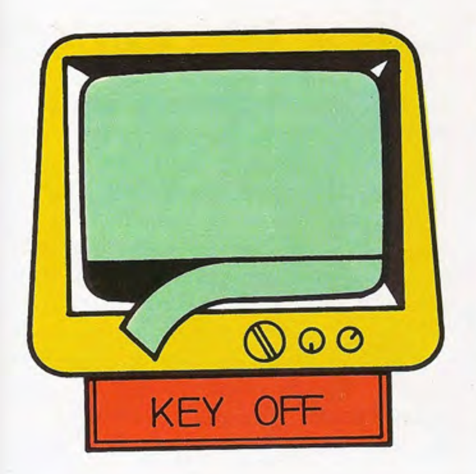
La línea 50 se encarga de dimensionar la matriz A. La dimensión que realmente se va a utilizar se concreta en 30×20 elementos; no obstante, es preciso añadir un elemento más, por dimensión, para evitar inconvenientes que surgirán más adelante.

Las restantes instrucciones de la rutina definen un bucle que comienza en la línea 70 y termina en la 110. Este bucle colocará una mina en cada una de las pasadas; por supuesto, de forma aleatoria. Dentro del bucle, las líneas 80 y 90 generan la posición de la mina; posición que se calcula a partir del número aleatorio entregado por la función RND. Dicho valor ha de someterse a un cambio de escala para que resulte un número comprendido dentro del margen de 1 a 20 en el caso de Y2, y de 1 a 30 en el caso de X2.

Por último, la línea 100 inspecciona la casilla de origen (1,1) para evitar que coloque en ella una mina. La referida línea 100 puede ocasionar dificultades en el caso de que un dialecto BASIC utilizado no incorpore la estructura IF/THEN/ELSE; en tal situación es recomendable cambiar la línea 100 por el par de instrucciones siguientes:

```
100 IF Y2=1 AND X2=1 THEN GOTO 110
105 A(X2,Y2)=1
```

Para delimitar el terreno de juego se utilizará en la pantalla una línea que defina su perímetro. De ello se encargarán las instrucciones que siguen:




*Si el ordenador que estamos usando es un IBM-PC o compatible, o un MSX, será necesario eliminar de la pantalla la línea inferior, que representa las funciones asignadas a las teclas de función. Para ello se emplea la orden KEY OFF.*




*Para el minado del campo se recurre al artificio de crear una matriz con tantos elementos como casillas pueda recorrer el personaje. Por supuesto, la colocación de las minas se realiza de forma aleatoria.*

```
120 REM TRAZADO DEL CONTORNO
130 LINE (7,7)—(248,168),,B
```

Esta última instrucción debe ser sustituida en cada caso por su equivalente en el ordenador que se utilice. En esencia, su misión es crear un rectángulo que delimite la superficie; la esquina superior izquierda de este rectángulo se encuentra en el punto 7,7 y la inferior derecha en el 248,168.

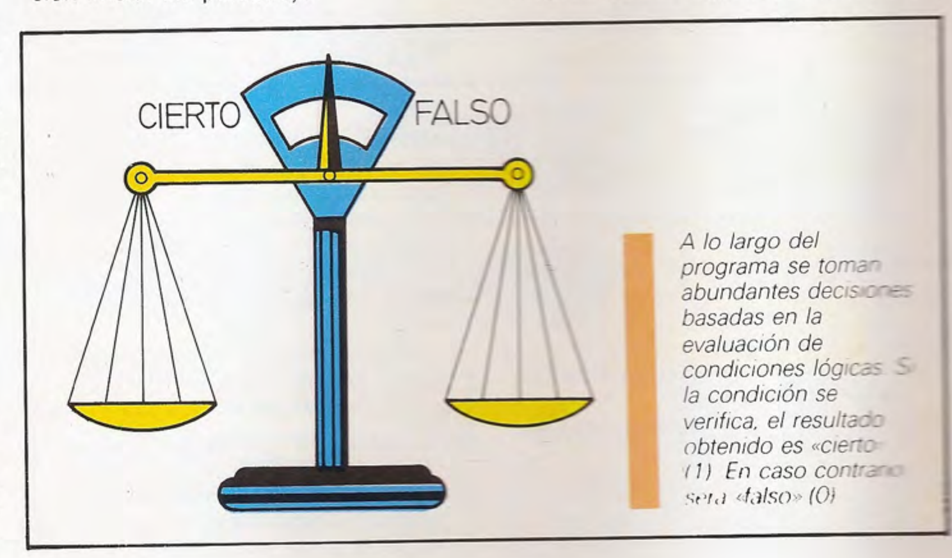
## Definición y movimiento del personaje

Tras ello, resulta conveniente inicializar las variables que definan la posición del jugador dentro del campo. Estas variables, coincidentes con X e Y contienen el valor actual de dicha posición. A su vez, las variables X1 e Y1 se emplean para conservar el valor de la posición que ocupaba anteriormente el personaje. La inicialización de estas variables exige precisar cuál será la posición inicial del personaje

Las instrucciones necesarias para esta operación pueden incluirse en una sola línea de programa, separando la inicialización de cada una de las variables por medio del signo dos puntos:

```
140 X=1 : Y=1 : X1=1 : Y1=1
```

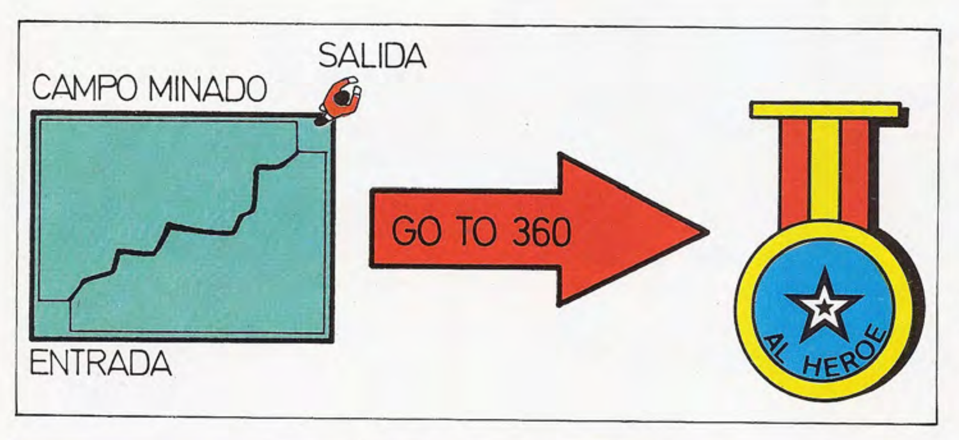
El desplazamiento a través del rectángulo de juego ha de controlarse desde el teclado. Para ello es necesario incluir una serie de instrucciones que permitan al programa detectar las pulsaciones que efectúe el usuario sobre el teclado. La función BASIC más adecuada para este cometido es INKEY$, puesto que no exige pulsar la tecla de retorno del carro para que el ordenador entienda que




*A lo largo del programa se toman abundantes decisiones basadas en la evaluación de condiciones lógicas. Si la condición se verifica, el resultado obtenido es «cierto» (1). En caso contrario será «falso» (0).*




Si el programa detecta la presencia de una mina en la misma casilla que el personaje, se producirá un salto a la rutina de salida asociada a «Derrota».




Si la travesía culmina con éxito, el programa bifurcará al punto de salida «victoriosa», y reflejará en la pantalla el número de movimientos realizados.

| TABLA DE FUNCIONES LÓGICAS | | | | | |
|---|---|---|---|---|---|
| Datos | | | | Funciones lógicas | |
| **A** | **B** | **A y B** | **A o B** | **no A** | **no B** |
| verdadero | verdadero | verdadero | verdadero | falso | falso |
| verdadero | falso | falso | verdadero | falso | verdadero |
| falso | verdadero | falso | verdadero | verdadero | falso |
| falso | falso | falso | falso | verdadero | verdadero |
| | | **AND** | **OR** | **NOT** | |

| TABLA DE VARIABLES | | | |
|---|---|---|---|
| A() | Matriz que almacena la posición de las minas. | I | Indice de bucle. |
| | | X,Y | Posición actual del personaje. |
| B$ | Variable que contiene el carácter pulsado. | X1,Y1 | Posición anterior del personaje. |
| CON | Contador de movimientos. | X2,72 | Coordenadas para la generación de las minas. |

el mensaje ha de ser aceptado y analizado:

```
160 B$=INKEY$: IF B$=" " TEHN GOTO 160
```

Por supuesto, la próxima operación a realizar consiste en analizar lo que el operador ha introducido. Llegados a este punto hay que estudiar qué teclas se utilizan para ordenar los movimientos a través del escenario del juego. Parece conveniente en este caso elegir teclas distribuidas de una forma lógica; por ejemplo, D, A, X y W.

El siguiente bloque de instrucciones se encargará de actualizar los valores de X e Y en función de la tecla que se pulse en cada instante:

```
170 IF X<30 THEN X=X—(B$="D")
180 IF X>1 THEN X=X+(B$="A")
190 IF Y<20 THEN Y=Y—(B$="X")
200 IF Y>1 THEN Y=Y+(B$="W")
```

Es muy posible que las operaciones situadas tras cada condición resulten un tanto extrañas. De hecho, no es muy ortodoxo mezclar la evaluación de condi-

ciones con instrucciones aritméticas; sin embargo, ello está permitido.

La evaluación de cada condición impuesta produce un resultado igual a cero si la condición es falsa, e igual a –1 cuando la condición resulta ser cierta. En este caso se aprovecha directamente el resultado de evaluar la condición para producir las modificaciones necesarias en las coordenadas que definen la posición del personaje.

Para controlar el número de movimientos que se realizan hasta terminar el juego se emplea una variable contadora: CON. Su valor se inicializa a cero en la línea 30 y se incrementa sucesivamente tras cada movimiento.

```
210 CON=CON+1
```

El siguiente paso consiste en colocar en la pantalla a nuestro personaje, representado por un asterisco, y borrar el asterisco de la posición que ocupaba anteriormente.

```
220 LOCATE Y1+1, X1+1
230 PRINT " "
240 LOCATE Y+1, X+1
250 PRINT "*"
```

Las instrucciones de las líneas 220 y 240 trasladan el cursor a la posición de pantalla adecuada, ya sea para imprimir un espacio en blanco encima de la posición que el personaje ocupaba anteriormente, o bien para visualizar el asterisco en la posición actual. La posición que anteriormente ocupaba el asterisco en la pantalla se almacena en las variables X1 e Y1. Para ello se asignan a estas variables los valores de X e Y, antes de actualizar estos últimos. Así, tras la actualización de X e Y, X1 e Y1 conservarán el contenido precedente de las variables X e Y. La instrucción adecuada para realizar esta asignación es la siguiente:

```
260 X1=X : Y1=Y
```

## ¿Exito o explosión?

Tras actualizar la posición, la próxima tarea que debe ocupar el programa es comprobar a dónde lleva el movimiento realizado. Al efecto, es necesario efectuar una serie de comprobaciones y tomar las oportunas medidas dependiendo de los resultados que se obtengan:

*Diagrama de flujo correspondiente al programa «Campo minado».*

COMIENZO
COLOCACION MINAS
CAPTACION DE UNA PULSACION
ACTUALIZACION POSICION
DIBUJO ASTERISCO
¿FIN?
SI
NO
¿MINA?
SI
NO
¿MINA CERCA?
SI
NO
PITIDO
DERROTA
FIN
VICTORIA
FIN

```
280 IF X=30 AND Y=20 THEN GOTO 360
```

La línea 280 comprueba si se ha alcanzado la esquina opuesta del campo minado, cuyas coordenadas son X=30 e Y=20. Si se ha llegado a dicho punto se producirá una bifurcación hacia la línea 360, donde se encuentra la rutina adecuada para tratar este supuesto:

```
290 IF A(X,Y)=1 THEN GOTO 340
```

Por su parte, en la 290 se comprueba si la posición que ocupa el personaje en este preciso instante coincide con una de las minas. De ser cierto el juego habrá terminado con una fatal consecuencia. De inmediato se bifurcará a una rutina situada a partir de la línea 340, la cual imprimirá el oportuno mensaje antes de poner fin al programa:

```
310 IF A(X+1,Y)=1 OR A(X—1,Y)=1 OR A(X,Y—1)=1 OR
    A(X,Y—1)=1 THEN BEEP
```

Esta línea de programa es la encargada de simular la actuación del detector de proximidad de minas. Genera un sonido a través del altavoz en el caso de

```
10 REM JUEGO DEL CAMPO MINADO
20 CLS:SCREEN 2
30 CON=0
40 KEY OFF
50 DIM A(31,21)
60 REM COLOCACION DE LAS MINAS
70 FOR I=1 TO 40
80 X2=1+INT(RND*30)
90 Y2=1+INT(RND*20)
100 IF X2=1 AND Y2=1 THEN GOTO 110 ELSE A(X2,Y2)=1
110 NEXT I
120 REM TRAZADO DEL CONTORNO
130 LINE (7,7)-(248,168),,B
140 X=1:Y=1:X1=1:Y1=1
150 REM ACTUALIZACION DE LA POSICION
160 B$=INKEY$:IF B$="" THEN GOTO 160
170 IF X<30 THEN X=X-(B$="D")
180 IF X>1 THEN X=X+(B$="A")
190 IF Y<20 THEN Y=Y-(B$="X")
200 IF Y>1 THEN Y=Y+(B$="W")
210 CON=CON+1
220 LOCATE Y1+1,X1+1
230 PRINT " "
240 LOCATE Y+1,X+1
250 PRINT "*"
260 X1=X:Y1=Y
270 REM COMPROBACION DE LA POSICION CON RESPECTO A LAS MINAS
280 IF X=30 AND Y=20 THEN GOTO 360
290 IF A(X,Y)=1 THEN GOTO 340
300 REM COMPROBACION DE PROXIMIDAD DE MINAS
310 IF A(X+1,Y)=1 OR A(X-1,Y)=1 OR A(X,Y+1)=1 OR  A(X,Y-1)=1 THEN BEEP
320 GOTO 160
330 REM RUTINAS DE FIN DEL JUEGO
340 PRINT "SE ACABO, HAS PERDIDO"
350 END
360 PRINT "GANASTE TRAS ";CON;" INTENTOS"
370 END
```

que se encuentre una mina en las proximidades de la posición que ocupa el personaje.

Por último, tan sólo queda retornar al comienzo de la rutina encargada de actualizar la posición:

```
320 GOTO 160
```

Al programa le faltan únicamente las dos rutinas de fin del juego: una a ejecutar en el caso de cruzar con éxito el campo minado, y otra que se ejecutará cuando el personaje tropiece con una de las minas. La rutina que tomará el control en caso de percance se reduce a dos líneas de programa:

```
340 PRINT "SE ACABO, HAS PERDIDO"
350 END
```

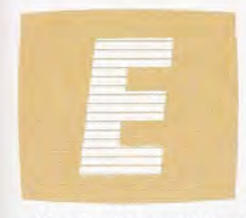
al igual que la asociada a un final victorioso:

```
360 PRINT "GANASTE TRAS ";CON;" INTENTOS"
370 END
```

Junto al texto se reproduce el listado completo del programa, salpicado por algunos comentarios (REM) que facilitarán la asimilación del mismo.

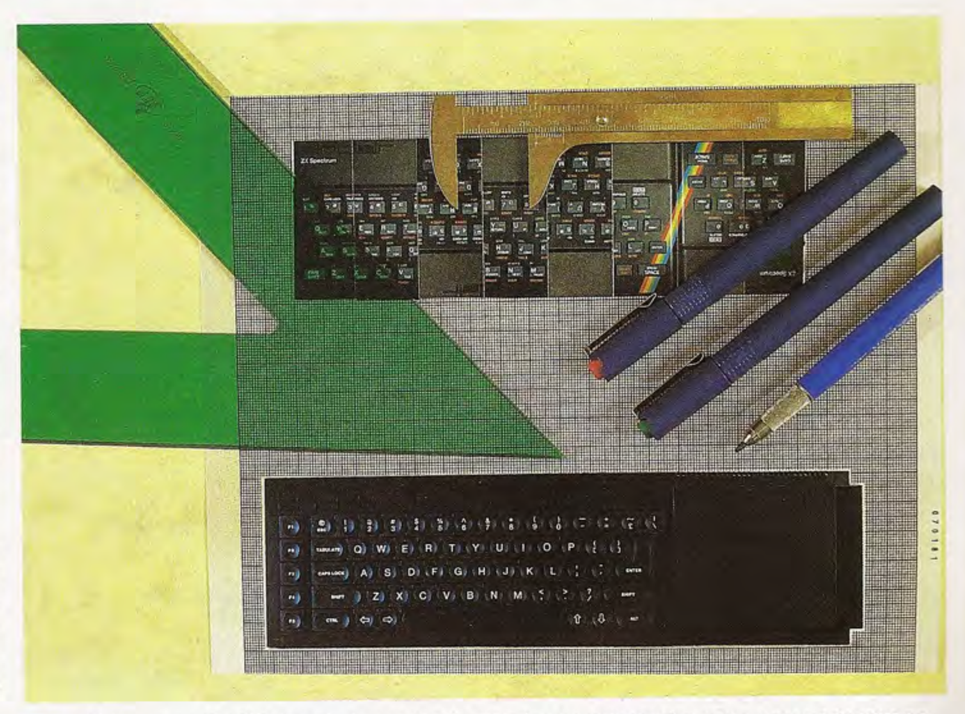
# El BASIC científico

## Estadísticas por ordenador



El empleo del ordenador en la sociedad actual se fundamenta en su gran potencia de cálculo. La mayor parte de las actividades en las que se hace necesario el uso del ordenador implican manipular un enorme volumen de datos. Para el manejo eficiente de grandes cantidades de datos es obligada una velocidad de cálculo también grande. Es ahí donde entra en juego el ordenador. Su capacidad de tratar muchos datos en poco tiempo lo convierte en una herramienta casi imprescindible.

En este capítulo se va a desarrollar un programa que hace uso de la capacidad anteriormente comentada. Se trata de un sencillo programa estadístico. Con él se pretende, a un tiempo, ejercitarse en la programación de tareas matemáticas y obtener un ejemplo de lo que se puede hacer con un ordenador.



*El lenguaje BASIC brinda todas las herramientas necesarias para crear programas de tipo científico o técnico ejecutables en ordenadores personales.*

## Estrategia de programación

A la hora de crear un programa hay que empezar realizando un análisis del problema a tratar. En este caso, se sentarán las bases de lo que se desea que haga el programa. En principio, la estadística trabaja con un número grande de datos. Esos datos serán introducidos por el operador para tomar parte, más tarde, en los cálculos pertinentes. Para un manejo más adecuado de los datos habrá que contar con variables de conjunto o arrays. Así pues, el conjunto de datos a analizar se almacenará en un array.

Una vez determinada la estructura de datos, el siguiente paso consiste en definir los procesos a realizar con los mismos. El proceso inicial ha de materializarse en la introducción de los datos. Tras esa introducción de datos se procederá al cálculo de los distintos resultados.

Los resultados que se van a conseguir son los siguientes:

— Número de datos introducidos.
— Suma de todos los datos.
— Valor máximo de los datos.
— Media aritmética.
— Varianza.
— Desviación estándar.

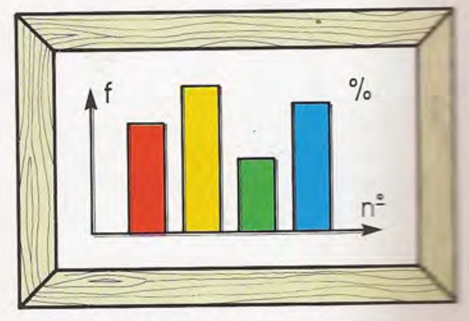
La última acción a realizar consistirá en la presentación de los resultados. De esta forma, el problema se subdivide en tres partes, de las cuales la segunda será la más compleja.

## Introducción de los datos

Antes de proceder a la introducción de los datos es necesario dimensionar la matriz que ha de contenerlos. Como ejemplo haremos uso de una matriz o array de 100 elementos. En el caso de desear un mayor número de datos, bastará con alterar esta primera línea del programa.

```
10 DIN A(100)
```

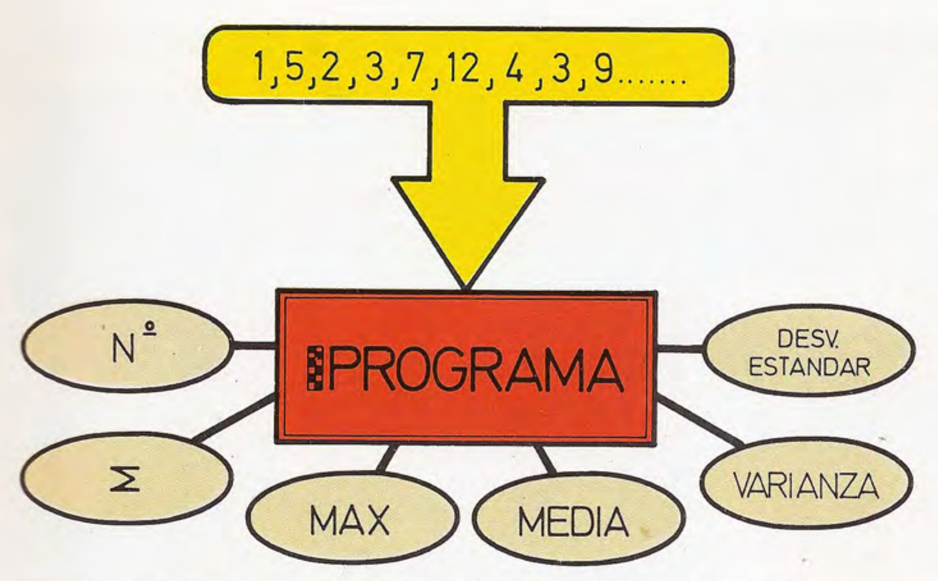
Para la introducción de los datos se empleará una instrucción INPUT reiteradamente. Ello se puede conseguir con el siguiente conjunto de líneas BASIC.



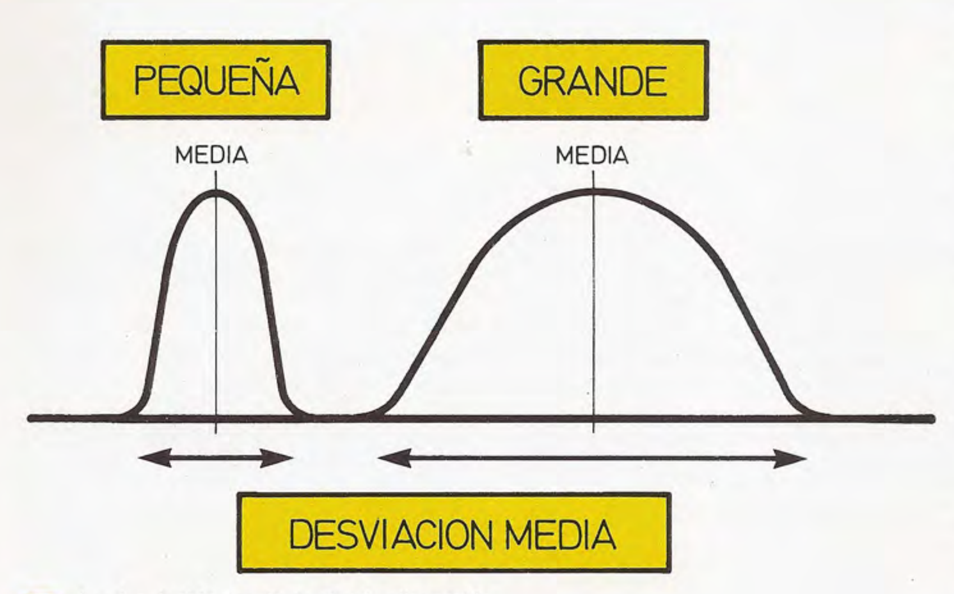

*El ordenador puede servir, entre otras cosas, para la realización de cálculos estadísticos.*

```
100 FOR I=1 TO 100
110 INPUT A(I)
120 NEXT I
```

Estas líneas permiten la introducción de los datos; sin embargo, obligan a teclear exactamente 100 números. Si lo que se desea es el análisis de una can-




*El programa, tras analizar los datos de entrada, proporciona una serie de parámetros que dan una idea de las características del conjunto.*




*La desviación media indica la dispersión de los datos alrededor de la media.*

tidad indeterminada de datos, las líneas mencionadas no servirán. Lo correcto en este caso sería ir introduciendo datos hasta finalizar, y una vez introducido el último dato indicar al ordenador que se ha completado la fase de entrada de datos. Esto se puede realizar de la siguiente forma:

```
100 LET I=1
110 INPUT B
120 IF B=99 THEN GOTO 200
130 LET A(I)=B
140 LET I=I+1
150 GOTO 110
```

Ahora se irán recogiendo los datos en la variable B. Esta será inspeccionada para ver si contiene el dato 99. Este dato es el que servirá para indicar la finalización de los datos. Si no se ha llegado al final, se pasa el valor de B a la matriz, se incrementa el índice I, y se vuelve a recoger otro dato.

Este método funciona correctamente, salvo por el hecho de que no permite introducir el dato 99 como uno más del conjunto a analizar. Esto se puede solventar de otra forma: utilizando como marca de final de datos un carácter no numérico. Ello hará que se complique el programa un poco más. En primer lugar, será necesario emplear una variable alfanumérica en lugar de B. A esta variable habrá de aplicársele una función que la convierta en dato numérico para su posterior tratamiento aritmético.

La solución más sencilla consiste en utilizar B$ en lugar de B y aplicarle luego la función VAL. El carácter que señale el fin de la introducción puede ser cualquiera no numérico, aunque aquí se utilizará la letra F (de fin). Este sería el aspecto de la nueva rutina:

```
100 LET I=1
110 INPUT B$
120 IF B$="F" THEN GOTO 200
130 LET A(I)=VAL(B$)
140 LET I=I+1
150 GOTO 110
```

## Cálculos

Tras la introducción de los datos, el programa ha de proceder al cálculo de los resultados. En este apartado se analizan las rutinas que proporcionan dichos cálculos.

El primer resultado a calcular es el número de datos introducido. Este valor será útil para el cálculo de otros parámetros. El número de datos se extrae directamente de la rutina de entrada. En ella se ha empleado un contador para variar el índice de la matriz A. Pues bien, al finalizar la introducción, dicho contador contendrá el número de datos que se han recogido. Así pues, la variable I contiene este primer dato. Para una mayor claridad del programa se traspasará ese dato a una variable específica que llevará el nombre de ND (número de datos):

```
200 LET ND=1
```

El resto de los resultados necesitan de unos cálculos más complejos que el expuesto. La suma de los datos se ha de realizar empleando un bucle. Los límites de ese bucle son los índices mayor y menor de la matriz. El menor va a ser siempre 1, ya que se empieza a rellenar desde el primer elemento. Sin embargo, el índice mayor dependerá de la cantidad de datos introducidos; o lo que es lo mismo: viene indicado por el recientemente calculado ND.

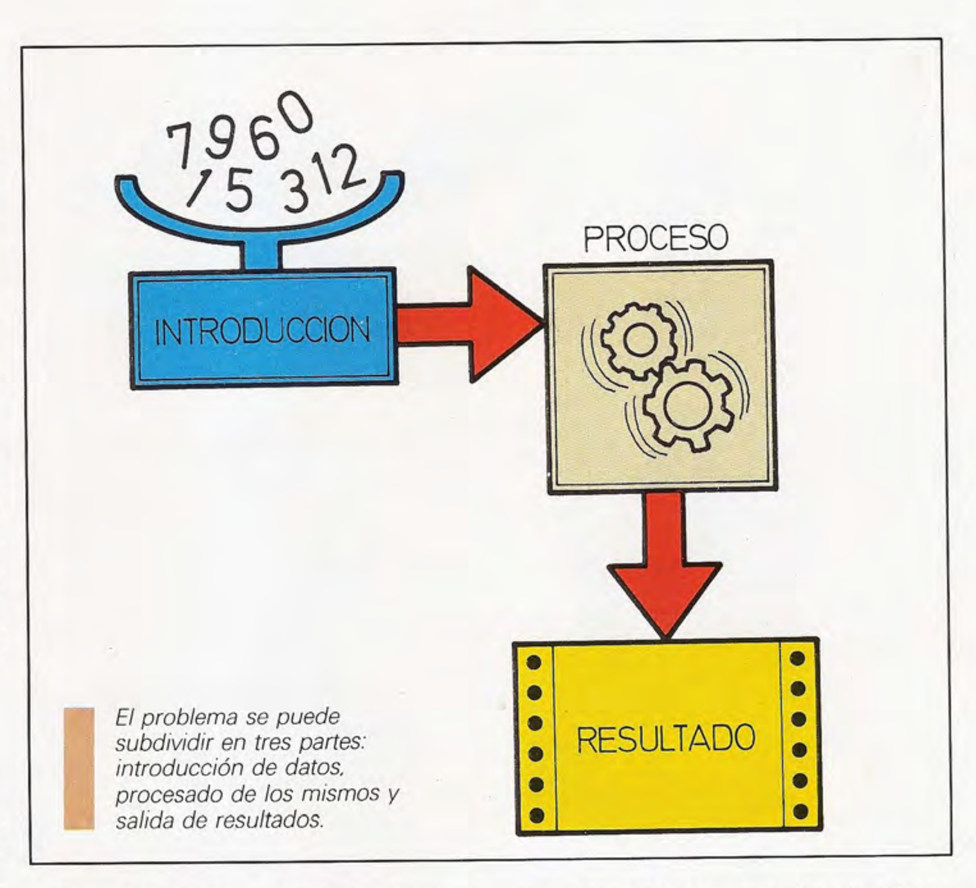
Una vez conocidos los límites se procede a diseñar el núcleo del bucle. Como se trata de hallar la suma, bastará con ir sumando cada dato en una variable que recoja así el total. Dicha variable recibirá el nombre de SUM. Y la línea que la actualice se limitará a sumar a SUM el valor del dato en curso:

```
310 FOR I=1 TO ND
320 SUM=SUM+A(I)
330 NEXT I
```

En este bucle se emplea de nuevo la variable I como contador. Ello no plantea ningún problema ya que su valor anterior ha sido almacenado en ND (línea 200). El uso de la misma variable en diferentes zonas del programa (y para usos bien distintos) proporciona un ahorro de memoria. Si se hubiera utilizado la variable J, por ejemplo, como contador en este bucle el ordenador habría reservado espacio para dos variables: I y J. Tal como se ha realizado aquí se ahorra el espacio correspondiente a la supuesta variable J.

La rutina de suma no está del todo finalizada. Es necesario cerciorarse de que se suman todos los datos y nada más que los datos. Que se suman todos los datos se comprueba por el hecho de que el índice I recorre todos los elementos introducidos en A. Para evitar que se sume algo más, es preciso verificar que SUM vale cero en la primera pasada del bucle. Si no fuera así, se estaría sumando una cantidad superflua al total. Así pues, la rutina completa quedaría de la siguiente forma:

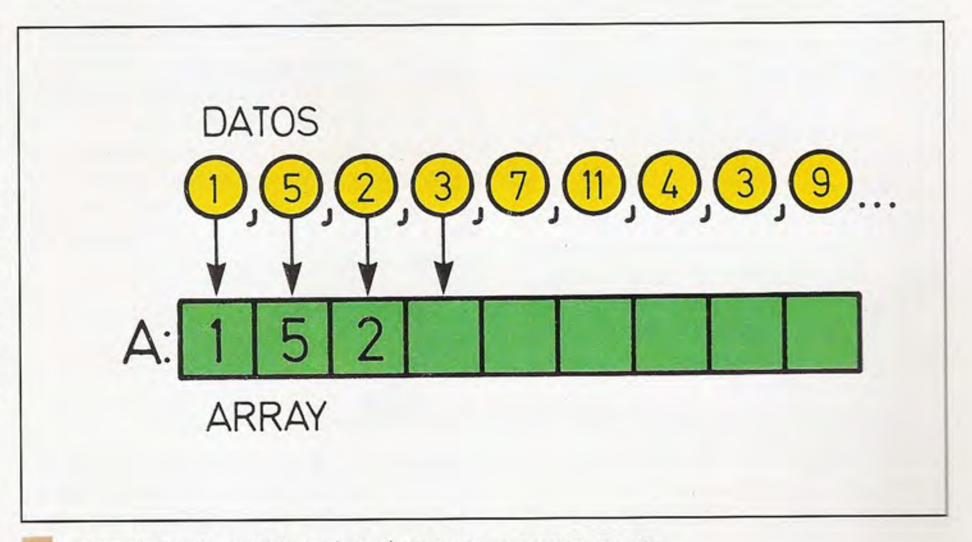
```
300 LET SUM=0
310 FOR I=1 TO ND
320 SUM=SUM+A(I)
330 NEXT I
```



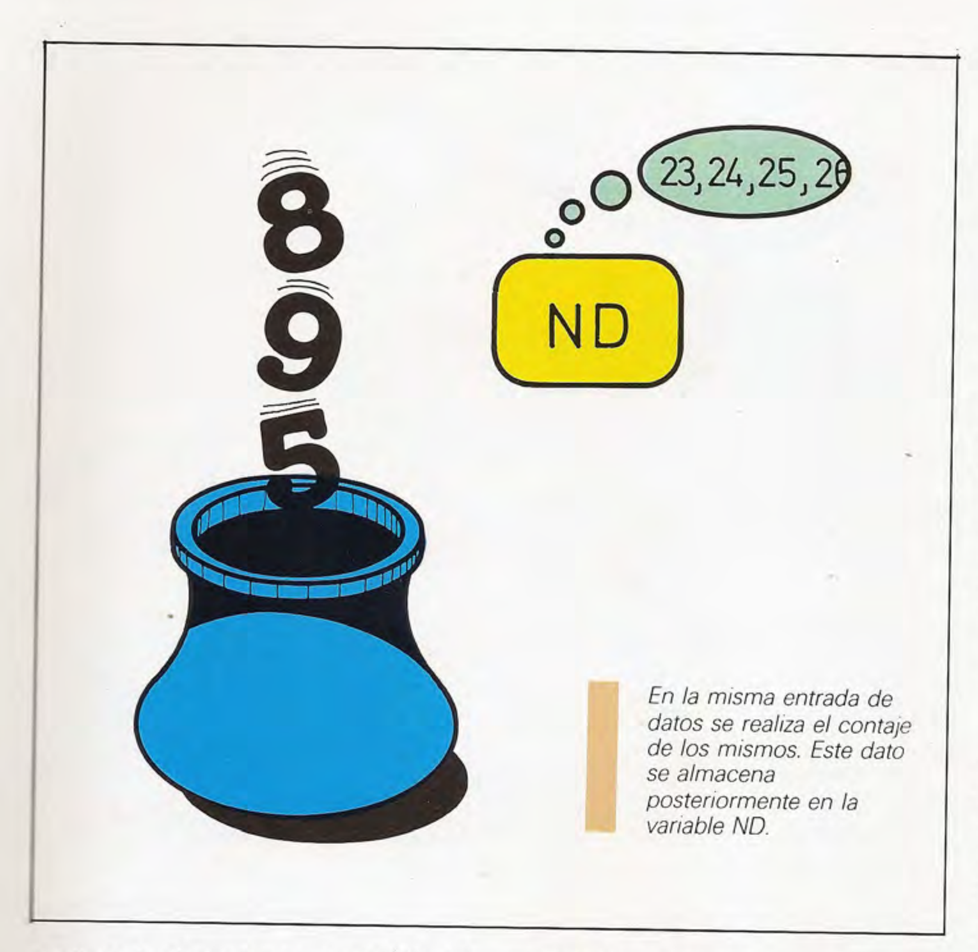

*El problema se puede subdividir en tres partes: introducción de datos, procesado de los mismos y salida de resultados.*

La ejecución de esta rutina dejará el valor de la suma de los datos en la variable SUM.

El siguiente resultado a calcular es el máximo de entre los datos proporcionados. La forma de hallar ese dato consiste en la comparación de los datos dos a dos. Tras cada comparación se retiene el mayor de los dos datos, y es ese dato retenido el que se compara con el siguiente. Tras recorrer todos los datos, no cabe la menor duda de que el dato arrastrado será mayor o igual que cada uno de los que componen el conjunto a




*Los datos de partida se introducen en un array para un manejo más cómodo por parte del programa.*




*En la misma entrada de datos se realiza el contaje de los mismos. Este dato se almacena posteriormente en la variable ND.*

analizar. La formulación en BASIC del método indicado es la siguiente:

```
410 FOR I=1 TO ND
420 IF A(I)>MAX THEN MAX=A(I)
430 NEXT I
```

Como en el caso anterior se ha hecho uso de nuevo de la variable I, ahorrándose así el espacio de una nueva variable.

En esta rutina puede surgir un problema: si el valor inicial de MAX es mayor que cualquiera de los del conjunto, se producirá un error de cálculo. En efecto, el máximo calculado no corresponderá a ninguno de los datos introducidos. Una primera solución consiste en la inicialización adecuada de la variable MAX, por ejemplo a cero. La rutina quedaría de esta forma:

```
400 LET MAX=0
410 FOR I=1 TO ND
420 IF A(I)>MAX THEN MAX=A(I)
430 NEXT I
```

Pero esta solución no resulta satisfactoria en la totalidad de los casos. Si da la casualidad de que todos los datos introducidos son negativos, el cero será mayor que todos ellos, reproduciéndose el error comentado. Por ello, la mejor solución sería inicializar MAX con un valor de los contenidos en la matriz. De esta forma no se introduciría la posibilidad de comparar con un valor inexistente dentro del conjunto a analizar. Se inicializará MAX con el primero de los datos introducidos. Esto significa que no es preciso comparar con el primero y, por lo tanto, el bucle puede comenzar en 2. De nuevo se muestra el aspecto de la rutina mejorada:

```
400 LET MAX=A(1)
410 FOR I=2 TO ND
420 IF A(I)>MAX THEN MAX=A(I)
430 NEXT I
```

Esta sería, pues, la rutina correcta. El mismo método puede emplearse para calcular el mínimo de los datos. En este caso se habrá de cambiar la comparación y, por supuesto, el nombre de la variable empleada. La rutina capaz de calcular el mínimo es la siguiente:

```
450 LET MIN=A(1)
460 FOR I=2 TO ND
470 IF A(I)<MIN THEN MIN=A(I)
480 NEXT I
```

Los siguientes resultados son los más puramente estadísticos. Para su cálculo se hace uso de algunos de los resultados ya hallados. El primero y más inmediato de los parámetros estadísticos de un conjunto de datos es la media. La media se define en base al número de datos y la suma de los mismos. Matemáticamente coincide con el cociente entre la suma y el número de datos. Su cálculo resulta, pues, muy sencillo partiendo de los datos que ya se conocen.

```
500 MED=SUM/ND
```

Otro parámetro importante es la desviación media. El valor medio (media) recién calculado indica el valor alrededor del cual se distribuye el conjunto de datos. Se llama desviación de un dato a la distancia que le separa de la media. Pues bien, la desviación media es la media de las desviaciones de los datos del conjunto. La desviación media proporciona una medida de la dispersión de los datos.

Para calcular la desviación media es preciso calcular las desviaciones individuales de los datos. Tras ello se ha de proceder al cálculo de la media de esos valores. Esto es, a sumarlos y dividir dicha suma por el número de datos. El cálculo de la desviación y la suma de ese valor al total puede realizarse de una sentada, en el interior del bucle que halla la suma total. He aquí la rutina apropiada:

```
550 LET SD=0
560 FOR I=1 TO ND
570 SD=SD+ABS(MED−A(I))
580 NEXT I
■
```

El cálculo de la desviación individual se realiza mediante la resta MED–AD(I). A ese valor se le aplica la función valor absoluto (ABS) para eliminar el signo, ya que el dato se puede encontrar tanto por encima como por debajo. Al dato así obtenido se le suma en la variable totalizadora SD (suma de desviaciones). En esta última variable quedará retenido el valor de la suma de las desviaciones. Para obtener la desviación media bastará con dividir el valor hallado por el número de datos, tal como se hace en la siguiente línea BASIC:

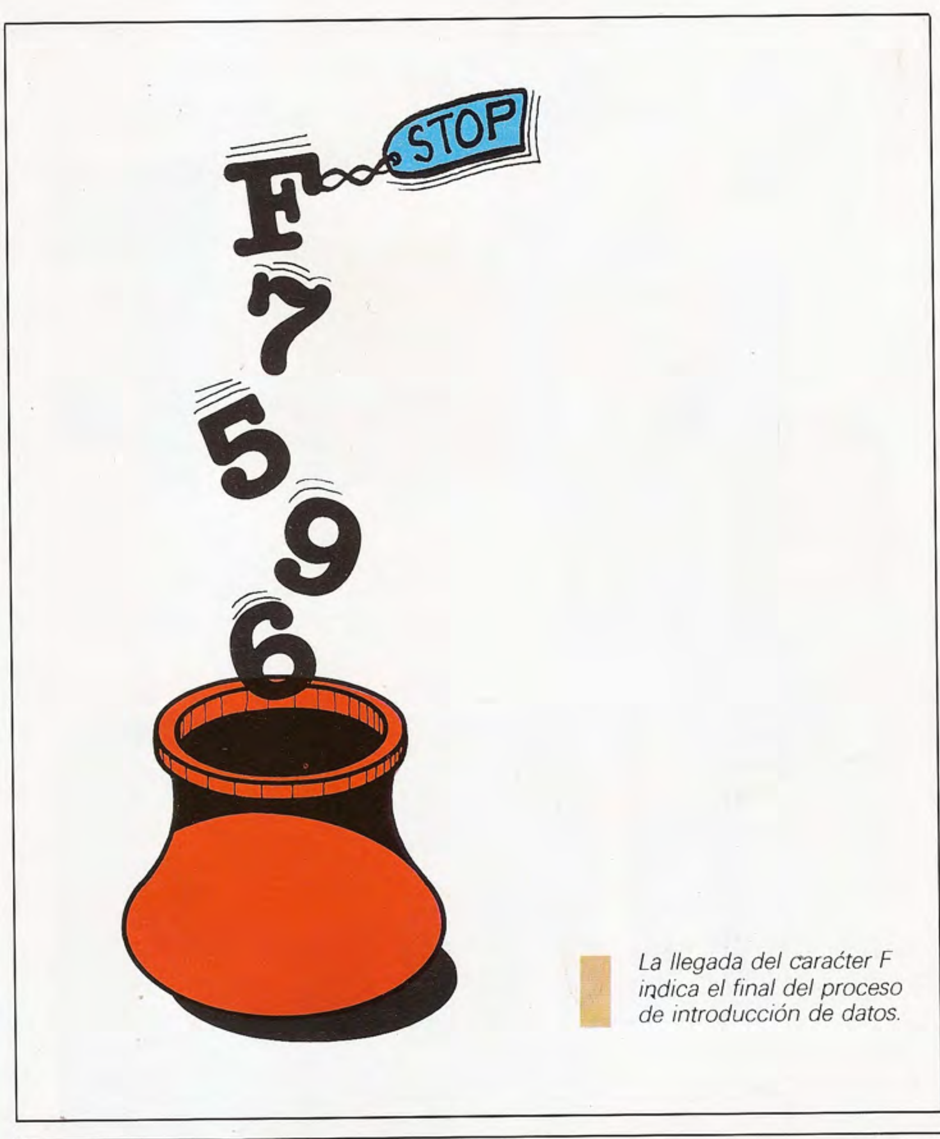
```
590 LET DM=SD/ND
```

La varianza es otro parámetro que da una idea de la dispersión de los datos. Sin embargo su cálculo es ligeramente diferente al de la desviación media. Lo que se halla en este caso no es la media de las desviaciones, sino la media de los cuadrados de las desviaciones. Esto requiere decir que la rutina de cálculo de la varianza será muy similar a la de la desviación media. En realidad el único cambio consiste en la sustitución de la función ABS por la operación de elevar al cuadrado la desviación. La rutina tomaría el siguiente aspecto:

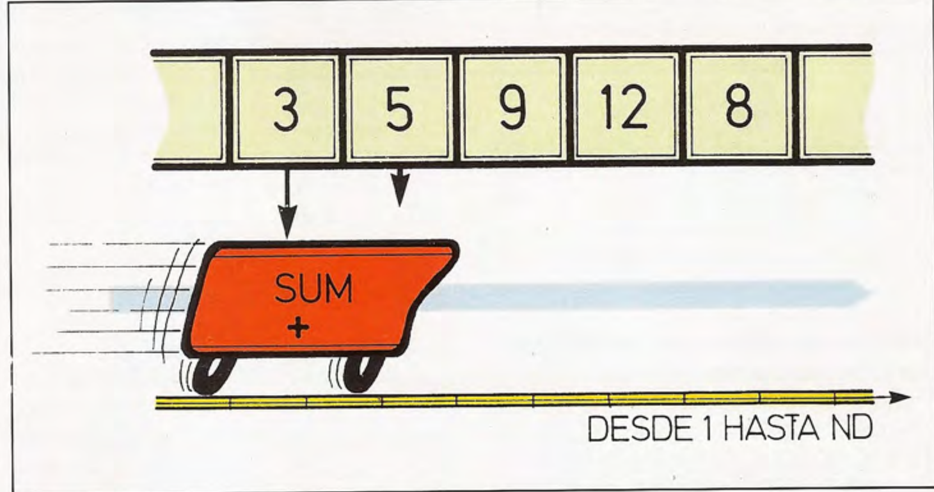
```
600 LET SD=0
610 FOR I=1 TO ND
620 SD=SD+(MED–A(I))2
630 NEXT I
640 LET V=SD/ND
█
```

Como se puede apreciar, se ha empleado la misma variable SD para almacenar la suma de los cuadrados. Ello no importa ya que el anterior valor de SD no es ya necesario, y en esta nueva rutina se inicializa de nuevo dicha variable.

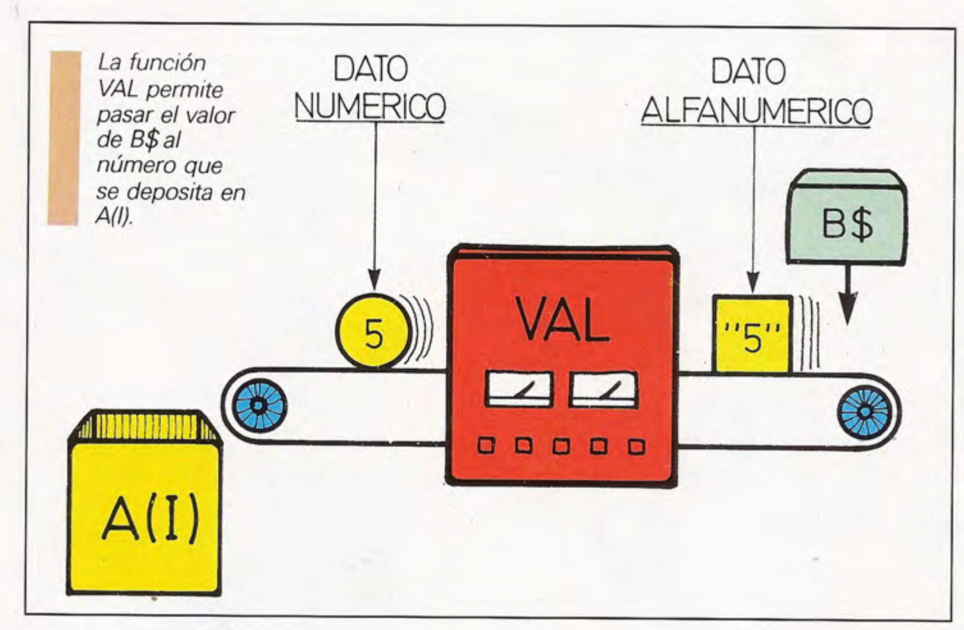
El último de los datos a calcular es la desviación estándar. Este valor se asemeja en gran manera a la desviación media. Si la varianza daba un valor basado en los cuadrados de las desviaciones (y no en las propias desviaciones,



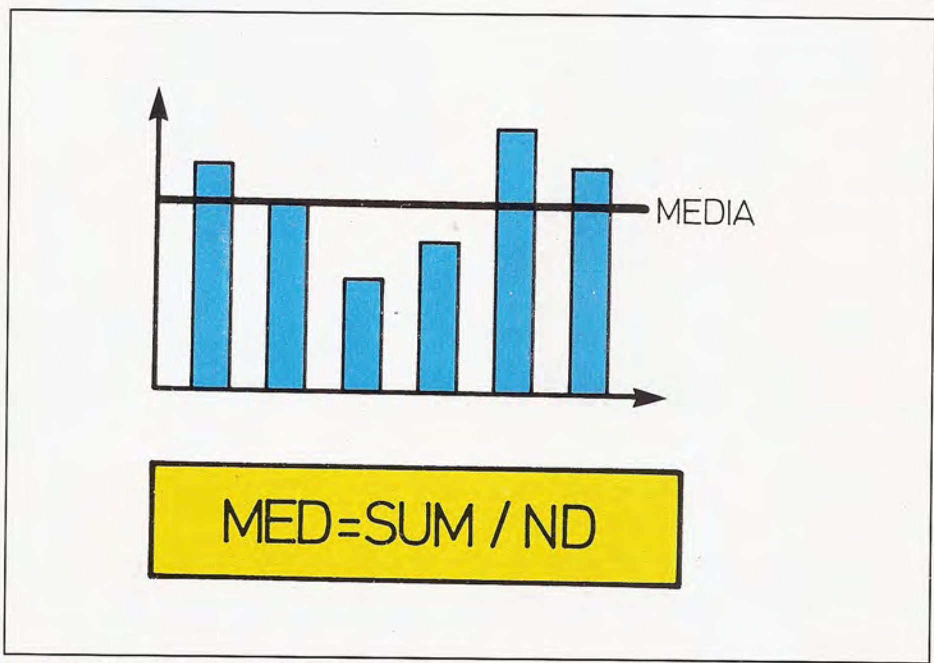

La llegada del carácter F indica el final del proceso de introducción de datos.




La suma de todos los datos se halla recorriendo la matriz desde el primero al último elemento. El total se almacena en SUM.




*La función VAL permite pasar el valor de B$ al número que se deposita en A(I).*




*Uno de los parámetros importantes es la media. Este dato se halla dividiendo la suma total entre el número de datos.*

como la desviación media) la desviación estándar corrige esa diferencia. La corrección consiste en extraer la raíz cuadrada del resultado ofrecido por la varianza. Así pues, la desviación estándar no es más que el resultado de hallar la raíz cuadrada de la varianza. Esto se codifica en BASIC con una sencilla línea:

```
650 LET DE=SQR(V)
```

Y con esto se completa la zona de proceso de los datos. En este terreno el programa calcula los resultados situándolos cada uno en una variable. El siguiente paso consiste en mostrar los resultados obtenidos. Este es el tema del siguiente apartado.

## Presentación de los resultados

A lo largo de toda la obra se ha comentado la importancia de una buena presentación de los datos. En el presente caso se dispone de los siguientes datos a representar:

— ND, SUM, MAX, MIN, MED, DM, V y DE

Estos datos se pueden representar en pantalla por medio del comando PRINT. La forma más sencilla de hacerlo es a través de un conjunto de líneas de este estilo:

(número de línea) PRINT "<nombre del resultado>: ";<variable>

En el argumento de PRINT se sitúa el comentario que identifica el dato a presentar. Tras esa cadena se mostrará el contenido de la correspondiente variable. Entre dato y dato convendrá dejar una línea en blanco, para dar una mayor claridad a la presentación. En conjunto, la rutina de salida de datos puede quedar de la forma siguiente:

```
1000 CLS
1010 PRINT
1020 PRINT "NUMERO DE DATOS: ";ND
1030 PRINT
1040 PRINT "SUMA DE LOS DATOS: ";SUM
1050 PRINT
1060 PRINT "DATO MAXIMO: ";MAX
1070 PRINT
1080 PRINT "DATO MINIMO: ";MIN
1090 PRINT
1100 PRINT "MEDIA ARITMETICA: ";MED
1110 PRINT
1120 PRINT "DESVIACION MEDIA: ";DM
1130 PRINT
1140 PRINT "VARIANZA: ";V
1150 PRINT
1160 PRINT "DESVIACION ESTANDAR: ";DE
```

Esta presentación puede reducirse con el uso de la particula AT en el argumento de PRINT. Ello, asimismo, permite colocar los datos numéricos en una misma columna. Con esa nueva facilidad la rutina quedaría como sigue:

```
1000 CLS
1020 PRINT AT(1,2) "NUMERO DE DATOS: "; AT(21,2) ND
1040 PRINT AT(1,4) "SUMA DE LOS DATOS: "; AT(21,4)
     SUM
```

```
1060 PRINT AT(1,6) "DATO MAXIMO: "; AT(21,6) MAX
1080 PRINT AT(1,8) "DATO MINIMO: "; AT(21,8) MIN
1100 PRINT AT(1,10) "MEDIA ARITMETICA: "; AT(21,10)
     MED
1120 PRINT AT(1,12) "DESVIACION MEDIA: "; AT(21,12)
     DM
1140 PRINT AT(1,14) "VARIANZA: "; AT(21,14) V
1160 PRINT AT(1,16) "DESVIACION ESTANDAR: ";
     AT(21,16) DE
```

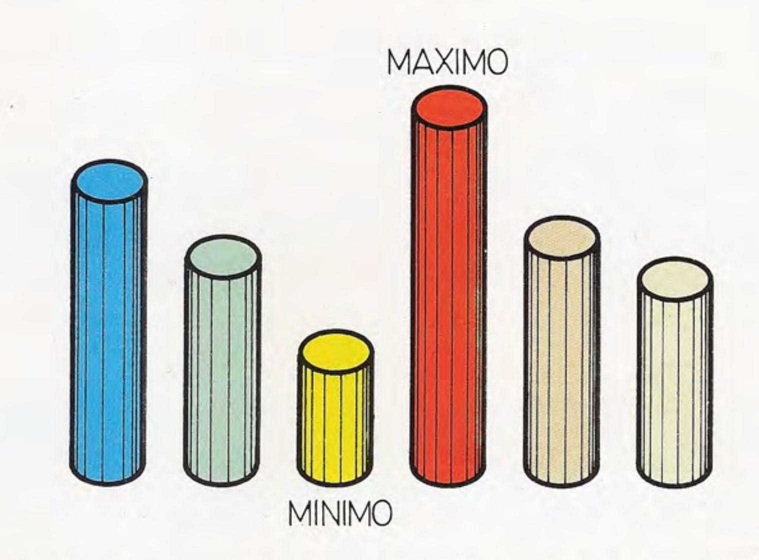
Los enunciados se colocan todos comenzando en la primera columna por medio de AT(1, Y). También se sitúan en la misma columna los datos; esta vez en la columna 21, siendo el argumento de AT (21,Y).

## Un complemento: salida por impresora

En programas de este tipo es frecuente la necesidad de apuntar los resultados obtenidos. Ello permite conservarlos y evita tener que ejecutar el programa cada vez que se desea volver a verlos. El método más sencillo de almacenamiento de los resultados consiste en plasmarlos en papel, a través de una impresora. Como complemento al programa expuesto se añade una rutina de impresión de los datos.

En principio, la rutina de impresión puede ser opcional. Así pues, el programa ha de preguntar al operador si desea esa impresión. Para ello se ha de incluir una rutina que recoja la respuesta del usuario. He aquí una posible rutina:

```
2000 PRINT
2010 PRINT "DESEA SALIDA IMPRESA (S/N)?"
2020 LET B$=INKEY$
2030 IF B$="N" THEN END
2040 IF B$<>"S" THEN GOTO 2020
2100 LPRINT "ANALISIS ESTADISTICO"
2110 LPRINT:LPRINT "LOS DATOS INTRODUCIDOS SON:"
2120 FOR I=1 TO ND
2130 LPRINT A(I);
2140 NEXT I
2150 LPRINT:LPRINT "RESULTADOS CALCULADOS:"
3010 LPRINT
3020 LPRINT "NUMERO DE DATOS: ";ND
3030 LPRINT
3040 LPRINT "SUMA DE LOS DATOS: ";SUM
3050 LPRINT
3060 LPRINT "DATO MAXIMO: ";MAX
3070 LPRINT
```




*Otros datos interesantes son el máximo y el mínimo de la distribución.*

NUMERO DE DATOS: 36
SUMA DE LOS DATOS: 360
MEDIA: 10
DESVIACION MEDIA: 3'62

*La presentación de los resultados se limita a la visualización de la cantidad calculada para cada parámetro. El usuario puede variar la rutina de visualización para eliminar el aspecto espartano que tiene la pantalla de la figura.*




*El programa permite la salida de resultados por impresora. Ello evita la necesidad de anotarlos a mano.*

```
5 REM ESTADISTICA
10 DIM A(100)
99 REM ENTRADA DE DATOS
100 LET I=1
110 INPUT B$
120 IF B$="F" THEN GOTO 200
130 LET A(I)=VAL(B$)
140 LET I=I+1
150 GOTO 110
199 REM NUMERO DE DATOS
200 LET ND=I
299 REM SUMA
300 LET SUM=0
310 FOR I=1 TO ND
320 SUM=SUM+A(I)
330 NEXT I
399 REM MAXIMO
400 LET MAX=A(1)
410 FOR I=2 TO ND
420 IF A(I)>MAX THEN MAX=A(I)
430 NEXT I
449 REM MINIMO
450 LET MIN=A(1)
460 FOR I=2 TO ND
470 IF A(I)<MIN THEN MIN=A(I)
480 NEXT I
499 REM MEDIA
500 MED=SUM/ND
549 REM DESVIACION MEDIA
550 LET SD=0
560 FOR I=1 TO ND
570 SD=SD+ABS(MED-A(I))
580 NEXT I
590 LET DM=SD/ND
599 REM VARIANZA
600 LET SD=0
610 FOR I=1 TO ND
620 SD=SD+(MED-A(I))2
630 NEXT I
640 LET V=SD/ND
649 REM DESVIACION ESTANDAR
650 LET DE=SQR(V)
999 REM PRESENTACION EN PANTALLA
1000 CLS
1020 PRINT AT(1,2) "NUMERO DE DATOS: "; AT(21,2) ND
1040 PRINT AT(1,4) "SUMA DE LOS DATOS: "; AT(21,4) SUM
1060 PRINT AT(1,6) "DATO MAXIMO: "; AT(21,6) MAX
1080 PRINT AT(1,8) "DATO MINIMO: "; AT(21,8) MIN
1100 PRINT AT(1,10) "MEDIA ARITMETICA: "; AT(21,10) MED
1120 PRINT AT(1,12) "DESVIACION MEDIA: "; AT(21,12) DM
1140 PRINT AT(1,14) "VARIANZA: "; AT(21,14) V
1160 PRINT AT(1,16) "DESVIACION ESTANDAR: "; AT(21,16) DE
1999 REM PRESENTACION EN IMPRESORA
2000 PRINT
2010 PRINT "DESEA SALIDA IMPRESA (S/N)?"
2020 LET B$=INKEY$
2030 IF B$="N" THEN END
2040 IF B$<>"S" THEN GOTO 2020
2100 LPRINT "ANALISIS ESTADISTICO"
2110 LPRINT:LPRINT "LOS DATOS INTRODUCIDOS SON:"
2120 FOR I=1 TO ND
2130 LPRINT A(I);
2140 NEXT I
2150 LPRINT:LPRINT "RESULTADOS CALCULADOS:"
3010 LPRINT
3020 LPRINT "NUMERO DE DATOS: ";ND
3030 LPRINT
3040 LPRINT "SUMA DE LOS DATOS: ";SUM
3050 LPRINT
3060 LPRINT "DATO MAXIMO: ";MAX
3070 LPRINT
3080 LPRINT "DATO MINIMO: ";MIN
3090 LPRINT
3100 LPRINT "MEDIA ARITMETICA: ";MED
3110 LPRINT
3120 LPRINT "DESVIACION MEDIA: ";DM
3130 LPRINT
3140 LPRINT "VARIANZA: ";V
3150 LPRINT
3160 LPRINT "DESVIACION ESTANDAR: ";DE
```

*PROGRAMA 1: Listado completo del programa «Estadística».*

```
3080 LPRINT "DATO MINIMO: ";MIN
3090 LPRINT
3100 LPRINT "MEDIA ARITMETICA: ";MED
3110 LPRINT
3120 LPRINT "DESVIACION MEDIA: ";DM
3130 LPRINT
3140 LPRINT "VARIANZA: ";V
3150 LPRINT
3160 LPRINT "DESVIACION ESTANDAR: ";DE
```

En esta rutina se ha añadido la impresión de los datos introducidos (líneas 2100 a 2140). Ello permite tener juntos datos y resultados. El resto de la rutina es, en esencia, idéntica a la de presentación en pantalla. La única variación se manifiesta en el uso de LPRINT en lugar de PRINT.

# Tablas de decisión

## Una técnica para dar eficacia a las tareas de programación



En el presente capítulo se pretende introducir al lector en una técnica de programación que le permitirá afrontar con mayor garantía y eficacia la confección de programas. Estas tablas empezaron a utilizarse en programación una vez que se comprobó que los tradicionales ordinogramas no eran suficientes en ciertos casos para resolver todas las situaciones que se presentaban.

La primera aplicación en la que se utilizaron en gran escala las tablas de decisión fue en el desarrollo de los lenguajes COBOL y FORTRAN, alrededor de 1958. La razón de su empleo residía en la presencia de una gran cantidad de decisiones frente a las que era necesario elegir un tratamiento.

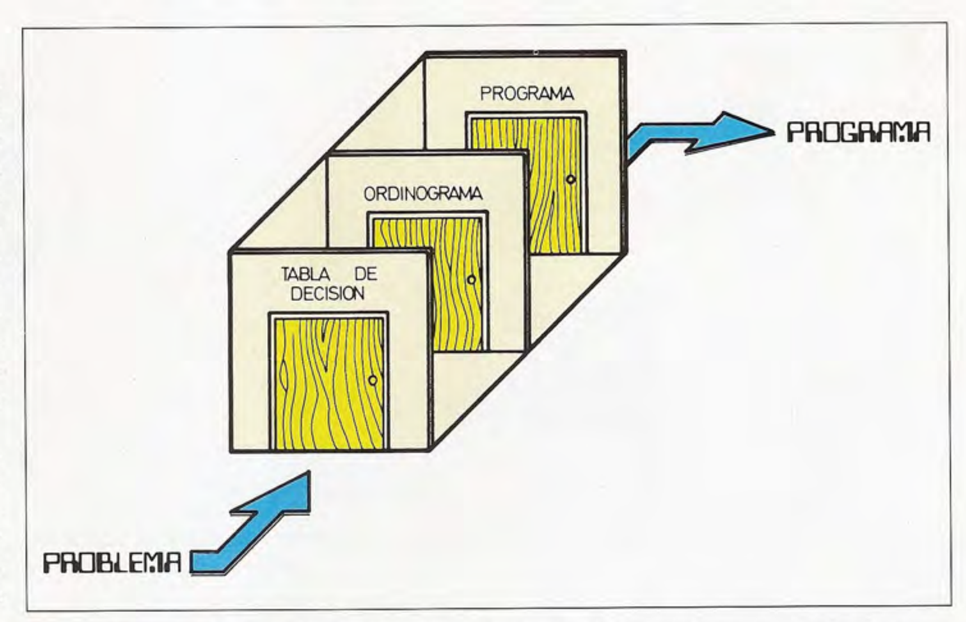
Las principales insuficiencias o limitaciones de los ordinogramas son las siguientes:

— Pueden presentarse varias soluciones para un mismo problema. Y todas ellas pueden ser perfectamente válidas.

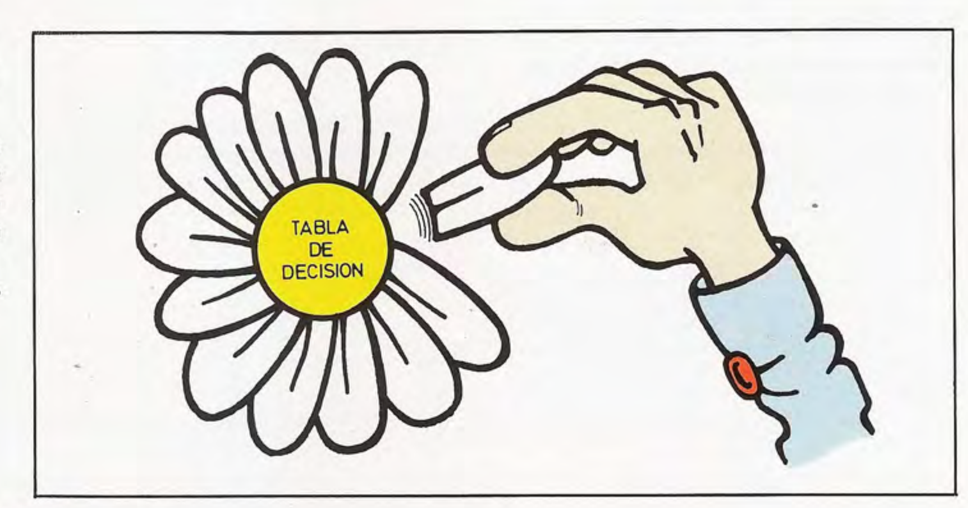
— Son difíciles de modificar. Y ésta es una necesidad frecuente, puesto que habrá que modificar el ordinograma cada vez que se altere el tratamiento a otorgar a las entradas que se presenten.

— Pierden eficacia en los lenguajes evolucionados.

— No son sistemáticos.




*Proceso que conviene seguir para el desarrollo eficaz y óptimo de un programa: creación de la tabla de decisión, ordinograma y programa.*




*Las tablas de decisión constituyen una técnica que aumenta la eficacia de todo programador.*

## Qué es una tabla de decisión

En la figura adjunta puede observarse la estructura típica de una tabla de decisión. Se observa que está dividida en cuatro zonas, representando cada una de ellas los siguientes conceptos.

1) *Condiciones que se pueden plantear*
Es un inventario o recopilación de todas las condiciones que intervienen en el proceso.

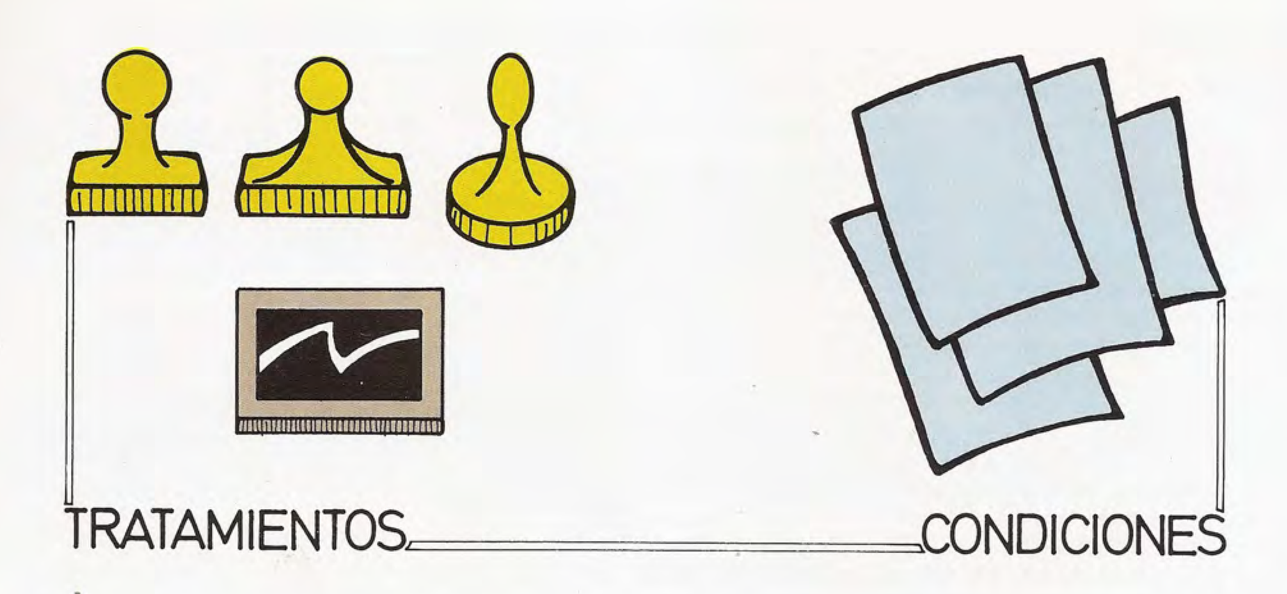
2) *Acciones o tratamientos*
En esta zona se han de incluir todos los posibles tratamientos o procesos que se deban llevar a cabo a partir de las condiciones impuestas.

3) *Entrada de condiciones*
Se incluyen todas las combinaciones posibles. En conjunto representa a todos los posibles casos que pueden surgir, incluyendo aquellos que no se debieran presentar en estricta lógica.

4) *Entrada de tratamientos*
Aquí se deben indicar todos los procesos o tratamientos a realizar. La selección de los tratamientos depende, evidentemente, de las condiciones que se presenten.

Antes de proseguir, es conveniente definir los conceptos que forman parte de la terminología asociada a las tablas de decisión.

SITUACION: designa a cada una de




*Una tabla de decisión relaciona los dos factores esenciales de toda alternativa: las condiciones de entrada y sus posibles tratamientos.*

las posibles combinaciones de valores que se pueden presentar en las entradas de condiciones.

REGLA DE DECISION: es el conjunto formado por la tabla de decisión y el tratamiento inherente a la misma.

ORDINOGRAMA 1

C2 — SI: C1 (SI: T1; NO: T1) — NO: C1 (SI: T2; NO: T1)

## Un ejemplo práctico

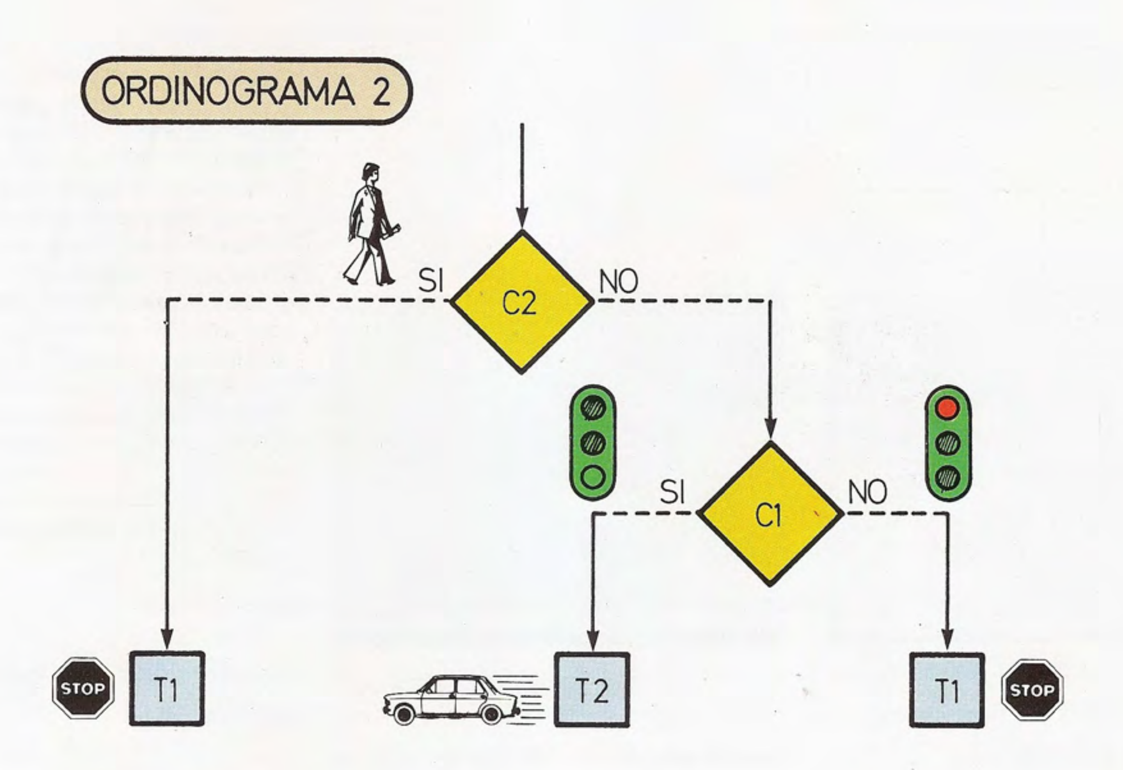
Suponga que pretendemos informar a un conductor sobre la acción que debe tomar al llegar a un cruce. La decisión se ha de basar en dos consideraciones: si el semáforo está verde y si se encuentra algún peatón en la calzada. Y las acciones a realizar son: detenerse en el caso de que el semáforo no esté en verde, o si se encuentra algún peatón en la calzada. Para empezar la construcción de la tabla, situaremos en ella las condiciones a partir de las cuales se ha de actuar (ver el desarrollo del ejemplo en el cuadro 1).

- C1: equivale a la condición de que el semáforo está en verde; las posibilidades son SI (verde) o NO (rojo).
- C2: indica la presencia o no de un peatón en la calzada; las posibilidades son SI o NO.

Tras ello se han de reflejar en la tabla las diferentes situaciones que pueden presentarse (S1 a S4). Estas quedan patentes por medio de grupos de respuestas a las condiciones (punto 2 del cuadro adjunto).

El siguiente paso (3) consiste en presentar en la tabla los diferentes tratamientos a realizar; en este caso son dos: continuar detenido, o poner el coche en marcha. Concretamente, T1 corresponde a permanecer detenido y T2 a poner el coche en marcha.

El siguiente paso es conectar las diferentes situaciones con los tratamientos a los que dan lugar. Para ello analizaremos cada una de las situaciones y,




*Aspecto final, una vez depurada la tabla de decisión, del ordinograma 1.*

a partir de las premisas del problema, deduciremos cuál es el tratamiento a otorgar.

La primera situación que se refleja en la tabla (S1) es «semáforo en verde» y «peatón cruzando». En tal caso, el coche ha de permanecer detenido para no arrollar al peatón.

La forma de reflejar este tratamiento en la tabla es poner un aspa en el punto donde se cruzan la columna que corresponde a dicha situación y la fila donde se encuentra el tratamiento a efectuar (ver punto 4 del cuadro adjunto).

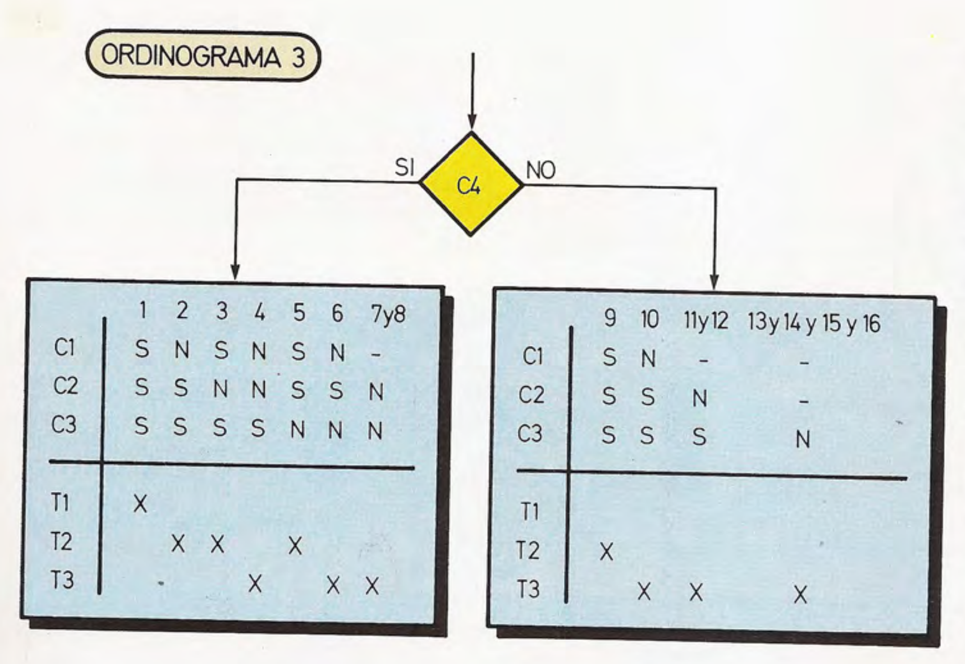
A continuación avanzaremos en el estudio de las restantes situaciones reflejando en las tablas los tratamientos oportunos hasta concluir la tabla de decisión correspondiente al ejemplo propuesto.

### TABLA DE DECISION

| | | | | | |
|---|---|---|---|---|---|
| C1 | S | N | ........... | S | S |
| C2 | N | S | ........... | S | N |
| : (1) : | : | : | (3) | : | : |
| CN | S | S | ........... | N | N |
| T1 | X | | | | |
| T2 | | X | | | |
| T3 | | | | X | |
| : (2) : | | | (4) | | |
| TM | | | | | X |

(1) CONDICIONES
(2) TRATAMIENTOS
(3) ENTRADA DE CONDICIONES
(4) ENTRADA DE TRATAMIENTOS

*Estructura de una tabla de decisión.*




| | 1 | 2 | 3 | 4 | 5 | 6 | 7y8 |
|---|---|---|---|---|---|---|---|
| C1 | S | N | S | N | S | N | - |
| C2 | S | S | N | N | S | S | N |
| C3 | S | S | S | S | N | N | N |
| T1 | X | | | | | | |
| T2 | | X | X | | X | | |
| T3 | | | | X | | X | X |

| | 9 | 10 | 11y12 | 13y14 y 15 y 16 |
|---|---|---|---|---|
| C1 | S | N | - | - |
| C2 | S | S | N | - |
| C3 | S | S | S | N |
| T1 | | | | |
| T2 | X | | | |
| T3 | | X | X | X |

## De tabla a ordinograma

Una vez confeccionada la tabla de decisión, da comienzo la siguiente fase: construir el ordinograma que permitirá codificar el programa adecuado.

En el caso que nos ocupa, el ordinograma asociado a la tabla de decisión es el que se reproduce bajo el título de ordinograma 1 (ver figura).

Al observarlo, se deduce de inmediato que dicho ordinograma no es óptimo. En efecto, la segunda decisión de la rama de la izquierda puede omitirse perfectamente, ya que en cualquiera de los casos conduce al mismo tratamiento.

Este hecho puede detectarse en la propia tabla de decisión (ver cuadro 2) y eliminar en ella tal redundancia.

Observando la tabla con más detalle, se descubre que las situaciones S1 y S2 conducen a un mismo tratamiento. Es obvio, pues, que para C2 verdadera (S) es posible prescindir del resultado de la condición C1; sea cual fuere su estado, siempre lleva al mismo tratamiento.

Esta situación, recibe el nombre de «indiferencia». En la tabla quedará reflejada según muestra el gráfico 2 del cuadro 2.

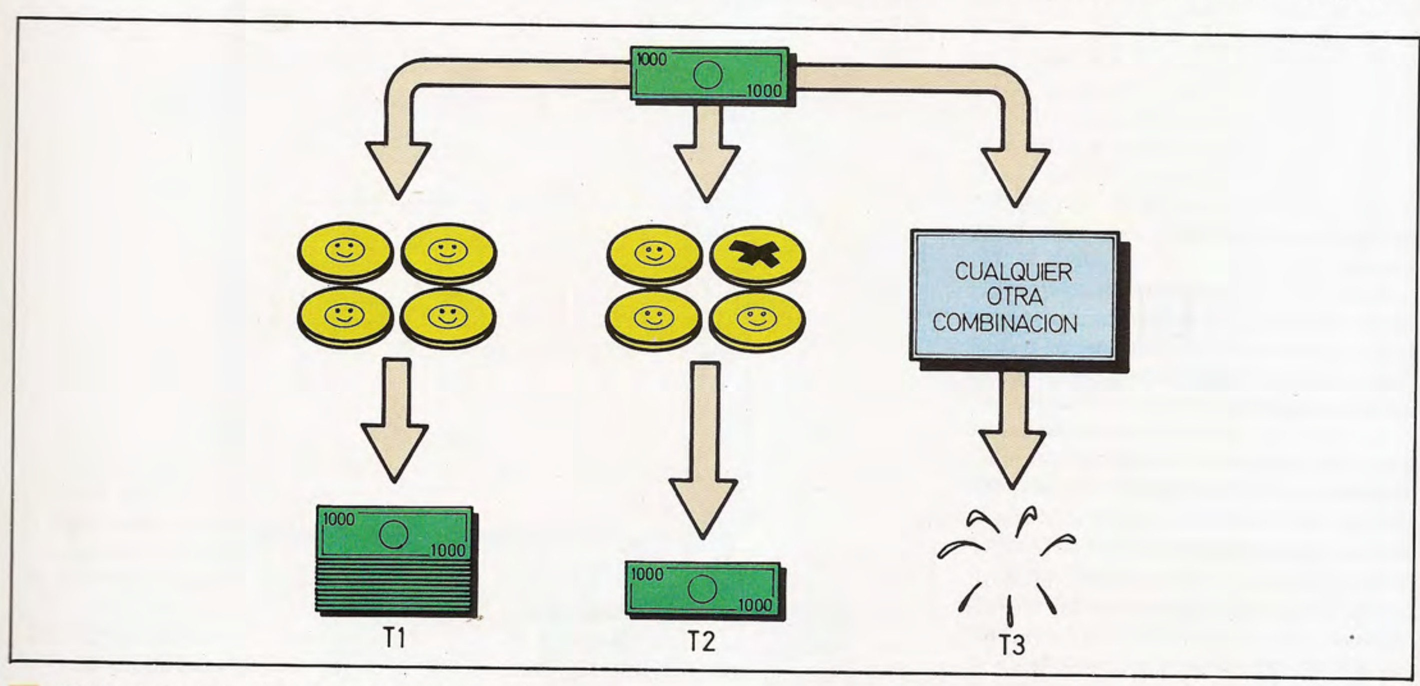
El nuevo ordinograma resultante aparecerá bastante más simplificado (ordinograma 2).

## Clasificación de las tablas de decisión

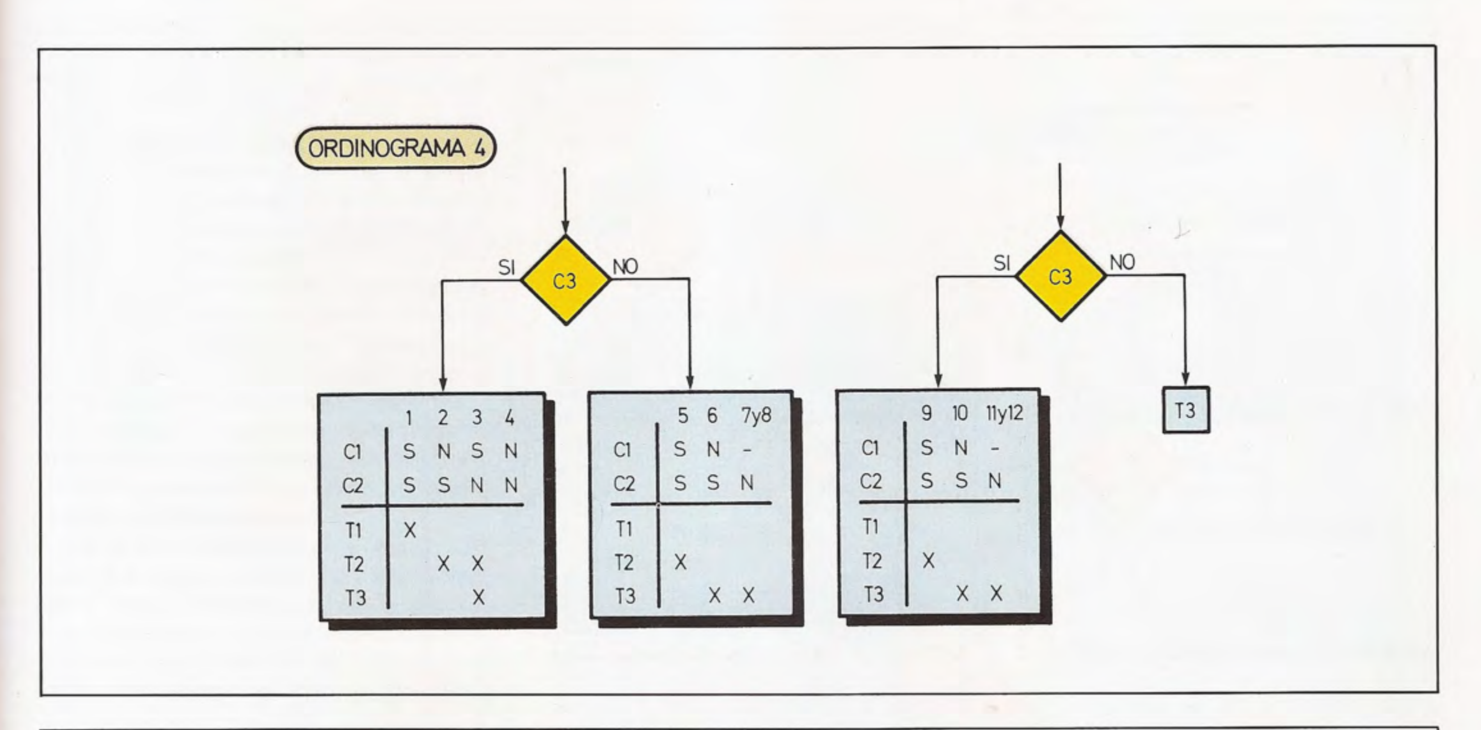
De acuerdo al tipo de condiciones que se planteen, las tablas de decisión pueden ser clasificadas en tres categorías:

- TABLAS LIMITADAS

En ellas las decisiones se pueden



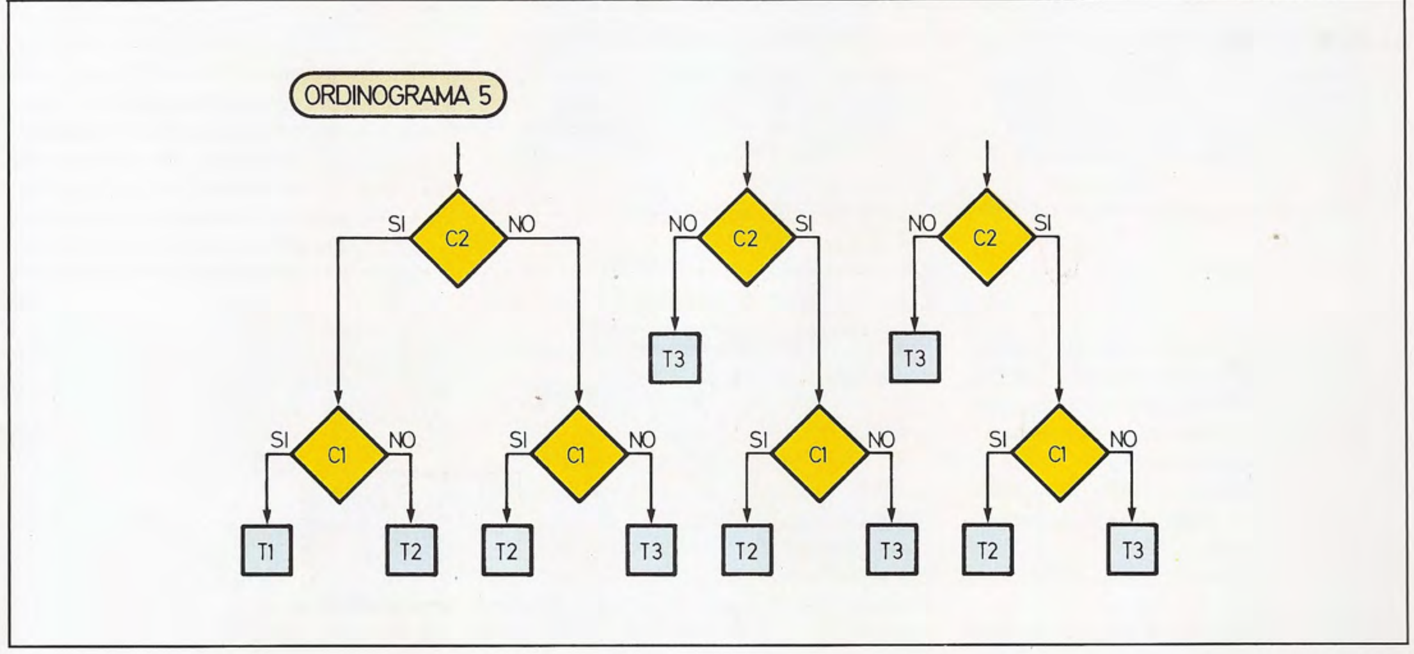

*Tratamiento de las distintas situaciones producidas al lanzar una moneda cuatro veces.*




| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| C1 | S | N | S | N |
| C2 | S | S | N | N |
| T1 | X | | | |
| T2 | | X | X | |
| T3 | | | X | |

| | 5 | 6 | 7y8 |
|---|---|---|---|
| C1 | S | N | - |
| C2 | S | S | N |
| T1 | | | |
| T2 | X | | |
| T3 | | X | X |

| | 9 | 10 | 11y12 |
|---|---|---|---|
| C1 | S | N | - |
| C2 | S | S | N |
| T1 | | | |
| T2 | X | | |
| T3 | | X | X |




plantear simplemente como que «se cumplen» o «no se cumplen»: un SI o un NO.

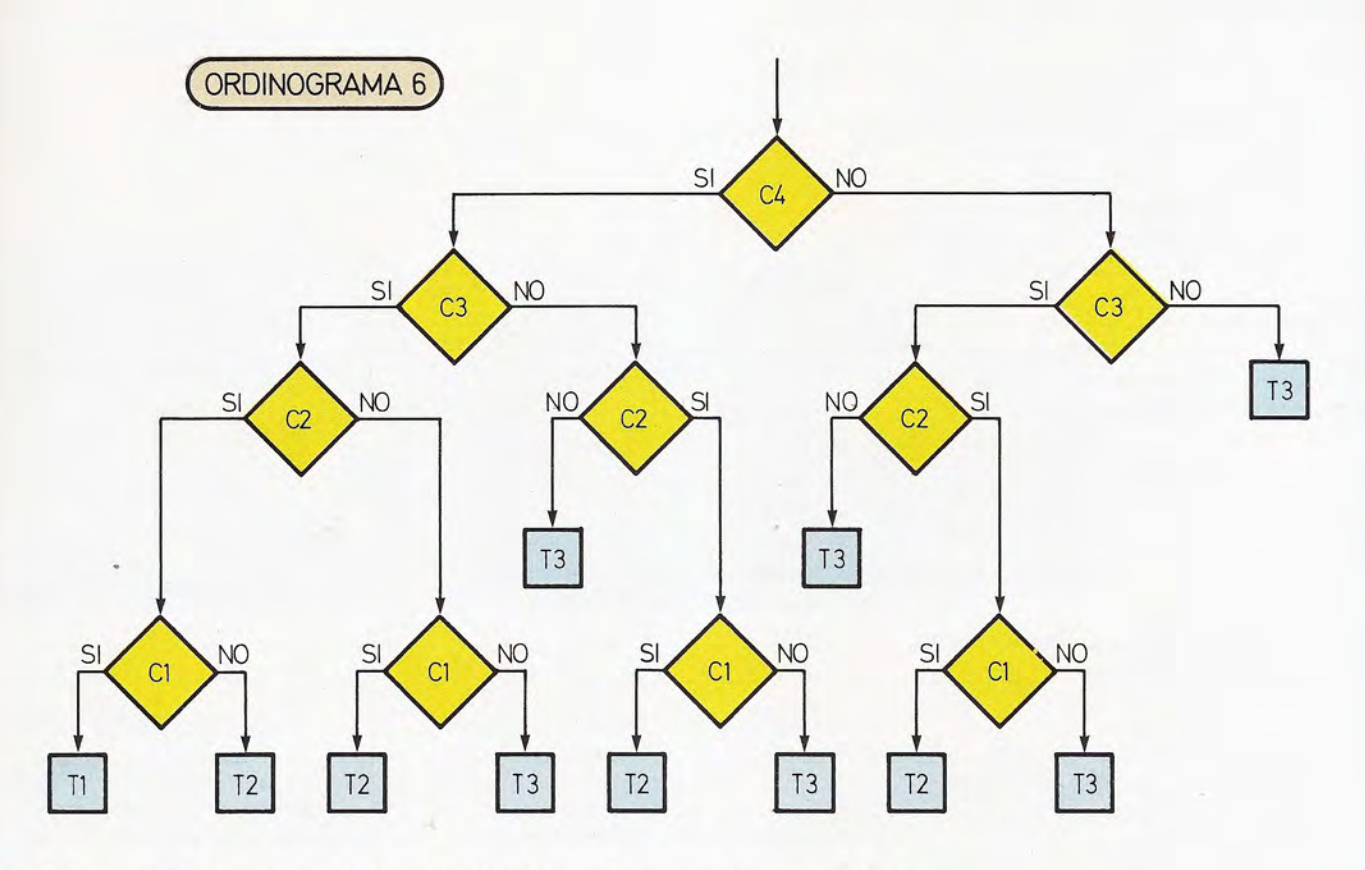
- TABLAS AMPLIADAS
  Son aquellas en las que las decisiones no se pueden plantear en los términos estrictos SI o NO, y es necesario recurrir a un acontecimiento de valores.
- TABLAS MIXTAS
  Evidentemente, esta categoría engloba a las tablas en las que aparecen mezcladas decisiones de ambos tipos.

## Un ejemplo evolucionado

Para ilustrar con todo detalle el uso de las tablas de decisión, vamos a desarrollar un ejemplo bastante más complejo que el utilizado al principio para introducir esta técnica.

Ahora, el número de posibles situaciones es bastante mayor, hecho que permitirá observar las facilidades que ofrecen las tablas de decisión a la hora de simplificar un problema complejo en el que intervienen un gran número de decisiones.

Se desea crear un programa que indique si se ha obtenido premio, y que revele su cuantía, en el siguiente juego: se lanza una moneda al aire 4 veces y




*Ordinograma final resultante de la asociación de los ordinogramas parciales 3, 4 y 5.*

se anotan los resultados; el jugador gana 20 veces la cantidad apostada si en las cuatro tiradas sale cara, recupera la apuesta si sale cara tres veces y pierde en cualquiera de las restantes situaciones. (La construcción de la tabla en sus distintas fases, puede observarse en el cuadro 3).

Por supuesto, se empezará definiendo las condiciones:

- C1: corresponde a la primera tirada; será SI en el caso de que haya salido cara.
- C2: segunda tirada de la moneda.
- C3: tercera tirada de la moneda.
- C4: cuarta tirada de la moneda.

Tal como establecen las reglas del juego, enunciadas en un párrafo anterior, son tres los posibles tratamientos a aplicar:

- T1: corresponde a la victoria del jugador; éste cobrará 20 veces la cantidad apostada.
- T2: reintegro, con lo que el jugador recuperará su inversión.
- T3: derrota del jugador.

En estas condiciones, la estructura de la tabla ofrecerá el aspecto señalado en la figura 1 del cuadro 3.

A continuación se trasladarán a la tabla todas las posibles combinaciones de condiciones para obtener el conjunto de situaciones que pueden darse en el juego (ver figura 2 del cuadro 3).

Tras ello hay que analizar cada una de las situaciones y decidir los correspondientes tratamientos (ver figura 3 del cuadro 3). Los tratamientos asociados a las diversas situaciones se marcarán con una X en las respectivas intersecciones de fila (situación) y columna (tratamiento).

Así, por ejemplo, la situación 3 (obtención de «cara» en las tiradas primera, tercera y cuarta) da lugar al tratamiento T2 (reintegro de la apuesta). También resulta obvio que la única situación que deriva en el tratamiento T1 (el jugador gana veinte veces el valor apostado) es la primera (cara en las cuatro tiradas).

Acto seguido dará comienzo la tarea de simplificar la tabla. Para ello, hay que observar qué pares de situaciones conducen a un mismo tratamiento; pares de situaciones que tan sólo se diferencian en el resultado de una de las condiciones.

Por ejemplo, situaciones como la siete y la ocho implican que no es necesario evaluar la condición C1, ya que el tratamiento de la situación, una vez que se han examinado las otras tres condiciones, es indiferente del valor que adopte dicha condición. Esta primera fase de simplificación da lugar a la tabla 4 incluida en el cuadro 3. Pero conviene revisar de nuevo las situaciones

por parejas, con el fin de comprobar si alguna de las parejas ya establecidas admite una nueva simplificación.

Es interesante hacer notar que las columnas correspondientes a las situaciones 13 y 14 y a la 15 y 16 son simplificables, ya que únicamente divergen en C2. Aunque no hay que acelerar la simplificación sin comprobar todos los extremos. Por ejemplo, la pareja de situaciones 7 y 8 no se puede simplificar con la 6, puesto que no sólo difieren en C2, sino también en la indiferencia de C1.

Finalmente, llegamos a la tabla 5 (cuadro 3); la tabla de decisión resultante a partir de la que se obtendrá el ordinograma asociado.

El ordinograma representativo del proceso debe ir resolviendo decisiones para llegar progresivamente a los correspondientes tratamientos. Es muy importante la elección del orden en el que estas decisiones han de ser tomadas. Un critero a aplicar es elegir para las primeras decisiones condiciones en las que no estén implicadas indiferencias.

En el caso que nos ocupa, existen dos condiciones a las que no afecta ninguna indiferencia; éstas son: C3 y C4. Por lo tanto, cualquiera de ellas es válida como primera condición a evaluar. Optemos, por ejemplo, por C4 como primera condición a testear (ver ordinograma 3).

Como se observa en la correspondiente ilustración, cada una de las ramas del ordinograma que parten de C4 ha dado lugar a una tabla de decisión específica, derivada de la primitiva. Algo semejante ocurre al desarrollar las ramas asociadas a las condiciones C3 (ordinograma 4); si bien, una de ellas ha desembocado en uno de los tratamientos (T3), debido a que no es necesario evaluar más condiciones.

Las otras tres ramas deben expandirse a partir de sus tablas respectivas, dando lugar a la distribución reflejada en el ordinograma 5.

Y ya sólo queda un paso más: enlazar las diversas secciones para construir el ordinograma final completo (ordinograma 6). Por supuesto, quedaría aún por acometer la fase de codificación, en la que se redactará el programa en un lenguaje informático, atendiendo a la estructura reflejada en el ordinograma.

**CUADRO 1**

1. Emplazamiento de las condiciones en la tabla.

| | | | | |
|---|---|---|---|---|
| C1 | S | N | S | N |
| C2 | S | S | N | N |

2. Definición de todas las posibles situaciones.

| | S1 | S2 | S3 | S4 |
|---|---|---|---|---|
| C1 | S | N | S | N |
| C2 | S | S | N | N |

3. Tratamientos que pueden realizarse.

| | S1 | S2 | S3 | S4 |
|---|---|---|---|---|
| C1 | S | N | S | N |
| C2 | S | S | N | N |
| T1 | | | | |
| T2 | | | | |

4. Tratamiento asociado a la situación S1.

| | S1 | S2 | S3 | S4 |
|---|---|---|---|---|
| C1 | S | N | S | N |
| C2 | S | S | N | N |
| T1 | X | | | |
| T2 | | | | |

5. Tratamiento de las distintas situaciones.

| | S1 | S2 | S3 | S4 |
|---|---|---|---|---|
| C1 | S | N | S | N |
| C2 | S | S | N | N |
| T1 | X | X | | X |
| T2 | | | X | |

*Secuencia de formación de la tabla de decisiones relativa al ejemplo del «vehículo» descrito en el texto.*

El ordinograma resultante presenta la sustancial ventaja de permitir la codificación del programa correspondiente de tal forma que éste presente la mínima ocupación posible de memoria. Esta es una característica inherente a los programas desarrollados a través de tablas de decisión perfectamente depuradas.

## CUADRO 2

*Depuración de la tabla de decisiones correspondiente al ejemplo del «vehículo» detallado en el texto.*

1. Tabla de decisión sin eliminar redundancias.

| | S1 | S2 | S3 | S4 |
|---|---|---|---|---|
| C1 | S | N | S | N |
| C2 | S | S | N | N |
| T1 | X | X | | X |
| T2 | | | X | |

2. Tabla resultante una vez simplificada.

| | S1 y S2 | S3 | S4 |
|---|---|---|---|
| C1 | - | S | N |
| C2 | S | N | N |
| T1 | X | | X |
| T2 | | X | |

## CUADRO 3

1. Condiciones y tratamientos.

| | |
|---|---|
| C1 | |
| C2 | |
| C3 | |
| C4 | |
| T1 | |
| T2 | |
| T3 | |

2. Definición de las posibles situaciones.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C1 | S | N | S | N | S | N | S | N | S | N | S | N | S | N | S | N |
| C2 | S | S | N | N | S | S | N | N | S | S | N | N | S | S | N | N |
| C3 | S | S | S | S | N | N | N | N | S | S | S | S | N | N | N | N |
| C4 | S | S | S | S | S | S | S | S | N | N | N | N | N | N | N | N |
| T1 | | | | | | | | | | | | | | | | |
| T2 | | | | | | | | | | | | | | | | |
| T3 | | | | | | | | | | | | | | | | |

3. Tratamientos asociados a las diversas situaciones.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C1 | S | N | S | N | S | N | S | N | S | N | S | N | S | N | S | N |
| C2 | S | S | N | N | S | S | N | N | S | S | N | N | S | S | N | N |
| C3 | S | S | S | S | N | N | N | N | S | S | S | S | N | N | N | N |
| C4 | S | S | S | S | S | S | S | S | N | N | N | N | N | N | N | N |
| T1 | X | | | | | | | | | | | | | | | |
| T2 | | X | X | | X | | | | X | | | | | | | |
| T3 | | | | X | | X | X | X | | X | X | X | X | X | X | X |

4. Resultado de la primera simplificación.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7y8 | 9 | 10 | 11y12 | 13y14 | 15y16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C1 | S | N | S | N | S | N | - | S | N | - | - | - |
| C2 | S | S | N | N | S | S | N | S | S | N | S | N |
| C3 | S | S | S | S | N | N | N | S | S | S | N | N |
| C4 | S | S | S | S | S | S | S | N | N | N | N | N |
| T1 | X | | | | | | | | | | | |
| T2 | | X | X | | X | | | X | | | | |
| T3 | | | | X | | X | X | | X | X | X | X |

5. Resultado de la segunda simplificación.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7y8 | 9 | 10 | 11y12 | 13y14y15y16 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| C1 | S | N | S | N | S | N | - | S | N | - | - |
| C2 | S | S | N | N | S | S | N | S | S | N | - |
| C3 | S | S | S | S | N | N | N | S | S | S | N |
| C4 | S | S | S | S | S | S | S | N | N | N | N |
| T1 | X | | | | | | | | | | |
| T2 | | X | X | | X | | | X | | | |
| T3 | | | | X | | X | X | | X | X | X |

*Secuencia de formación de la tabla de decisiones asociada al ejemplo «Lanzamiento de monedas».*

# El contable en casa

## Un programa de gestión, paso a paso



Para concluir la zona de la obra dedicada al lenguaje BASIC, avanzaremos un peldaño más en el terreno práctico aplicando los comandos estudiados hasta el momento. En esta ocasión, el programa es catalogable como de gestión.

Para poder construirlo de forma que tenga una cierta compatibilidad con cualquier tipo de ordenador personal, no se van a utilizar comandos «extraños», sólo disponibles en muy pocos dialectos BASIC. Al efecto, se utilizarán los comandos más corrientes en los intérpretes de BASIC de mayor difusión. La construcción del programa se complementará con explicaciones exhaustivas sobre la misión de cada comando y subrutina utilizadas.




*Un simple ordenador doméstico puede convertirse en el centro de contabilidad doméstica con el programa que presentamos en el presente capítulo.*

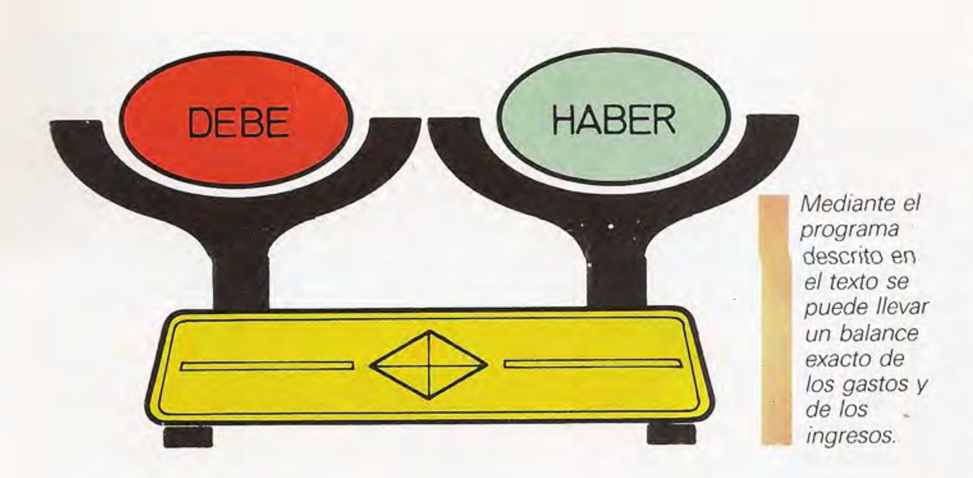
### El contable en casa

El programa que se va a desarrollar a continuación es un ejemplo de un sencillo banco de datos, destinado a almacenar información relativa a los pequeños o grandes gastos que se realizan en todo entorno familiar, o incluso en el de una pequeña empresa. Se trata de simular un "libro mayor", en el que se irán anotando cuidadosamente todos los gastos y los ingresos que se realicen en la familia o en la empresa, consignándolos en el lugar adecuado (en el «debe» o en el «haber»). El programa permite obtener información relativa a saldo actual en cada momento. Sobre esta base es posible añadir cualquier tipo de subrutina al programa, para calcular o ejecutar la acción o accciones particulares que necesite cada usuario, personalizando así su programa.

Para no hacer pesada la codificación, sólo se va a añadir una utilidad «extra», dejando a la atención del lector la adición de las nuevas utilidades que desee; ello resultará muy sencillo ya que bastará con incluir nuevas subrutinas que se pueden llamar por medio de un menú. Este será precisamente el modo en que estará organizado el programa que se va a codificar a lo largo de los




*Al acabar el tema dispondremos de un eficaz contable, encargado de llevar el libro de registro familiar.*




*Mediante el programa descrito en el texto se puede llevar un balance exacto de los gastos y de los ingresos.*

próximos párrafos. La utilidad «extra» que se va a incluir, permitirá llevar en paralelo con el saldo real, el saldo de nuestra cuenta corriente o cartilla de ahorros; como es sabido, los bancos tardan un tiempo en confirmar las operaciones que realizamos, y en consecuencia, hay momentos en los que no se sabe con precisión la cantidad disponible en la cuenta corriente. En cada apunte realizado se podrá indicar si éste ha sido o no confirmado por el banco: así mismo, también se podrá modificar el apunte una vez que el banco nos da la confirmación.

Una vez definido por encima el objetivo, es hora de poner los pies en tierra y comenzar a dar especificaciones prácticas del programa.

El principal elemento del programa será el fichero en el que se van a almacenar todos los datos que se introduzcan. Utilizar un fichero tipo aleatorio ofrece una serie de ventajas que serían muy largas de enumerar aquí, pero, a la vez, su manejo exige un mayor cuidado, hasta el punto de que la codificación de un programa que maneja ficheros aleatorios puede resultar muy complicada. Por esta razón, y para simplificar la labor, se utilizará un fichero secuencial que proporciona las características suficientes para nuestro propósito y que, además, será posible grabarlo en disco flexible o en casete; esto último supone una ventaja para aquellos usuarios que no dispongan aún de una unidad de disco.

El ficheros de datos que necesitamos, debe ser fácilmente manejado por el ordenador. La solución más sencilla es utilizar una matriz alfanumérica como fichero interno, ya que el empleo de subíndices facilita su manipulación. Como novedad, no se utilizarán esta vez matrices multidimensionales, sino que se empleará una única matriz alfanumérica unidimensional, en la cual se almacenarán por secciones cada uno de los datos introducidos. Para ello hay que dividir imaginariamente la matriz en sectores y campos, de tal forma que después se pueda localizar fácilmente la información.

Adoptando la terminología contable, se llamará asiento a cada sector de la matriz; a su vez, cada asiento se dividirá en campos de tamaño distinto para cada uno de los datos. Así, por ejemplo, una posible división de los campos podría ser la siguiente:

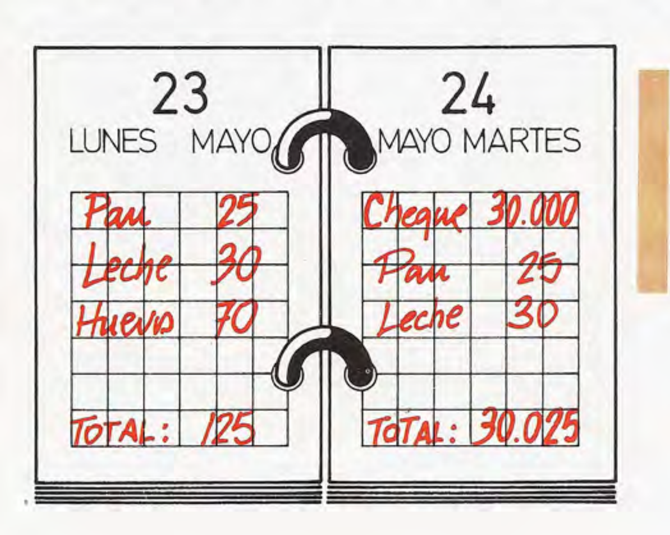
— Un campo de 1 carácter para la confirmación por el banco.

— Un campo de 6 caracteres para la fecha.

— Un campo de 20 caracteres para la descripción del apunte realizado.

— Un campo de 1 carácter para el signo del cargo (debe/haber).

— Y, finalmente, un campo de 8 caracteres para el importe.




*Llevando un control diario de asientos es posible obtener en cualquier momento el estado de nuestra economía.*

Todos estos campos se han distribuido en su longitud suponiendo que se dispone de una pantalla de 24 filas y 40 columnas. Con ello, cada sector o asiento del fichero tendrá 40 espacios de longitud, lo que pemitirá imprimirlo directamente en la pantalla. Para definir el tamaño de la matriz que necesitamos sólo queda por establecer el número de asientos que se quieren utilizar. Si este número lo fijamos en 500, ya se puede dimensionar la matriz y con ello crear las primeras líneas del programa:

```
10 DIM    DATO$(501*40),M$(40),FECHA(6),
   CON$(20),IMP$(8),CARGO$(1),VALE$(1)
20 DATO$=" ":DATO$(501*40)=" ":DATO$(2)=DATO$
30 LET A=1:LET PA=1:LET SR=0:LET SC=0:LET P=
40 OPEN # 1,4,0,"K"
```

Las líneas 10, 20 y 30 se encargan de dimensionar e inicializar las diferentes matrices variables que serán necesarias a lo largo del programa. Así, DATO$() consituye el ya mencionado fichero interno de datos que se ha dimensionado con el valor 501*40, para poder incluir en su último sector la marca de fin de datos.

El cometido de las restantes variables se detalla a continuación. M$ es una variable de utilidad general adecuada para la obtención de datos del teclado, impresión de mensajes en pantalla, etc. FECHA() se utilizará para almacenar temporalmente la fecha del asiento antes de introducirla definitivamente en el fichero, por si hay algún error y es preciso modificarla. Con el mismo fin se utilizarán las siguientes matrices: CON$(), IMP$(), CARGO$() y VALE$(), para los campos: concepto, importe, cargo (debe="D"/haber="H") y confirmación del banco, respectivamente.

La línea 20 se encarga de inicializar el fichero de datos con caracteres blancos (" "); si no funcionara en su ordenador puede sustituirla por un bucle FOR/NEXT como el siguiente:

```
20 FOR I=1 TO 501*40:DATO$(I)=" ":NEXT I
```

Las restantes variables numéricas que son inicializadas en la línea 30 son las siguientes: A para el número de asiento (inicialmente=1); PA, para el primer número de asiento de una página, ya que al no ser posible visualizar a la vez los 500 asientos del fichero éstos se consultarán por páginas; SR para el saldo real; SC para el saldo de la cuenta bancaria y P como puntero de la línea de la página visualizada, para introducir la confirmación por el banco del asiento señalado.

Por último, la línea 40 se encarga de abrir un canal de entrada para el teclado, por el que se introducirán todos los datos. Respecto a esta línea hay que decir que su formato varía mucho de unas máquinas a otras, por lo que será preciso acudir al manual BASIC de cada máquina. Si en algunos casos no fuera posible utilizar esta técnica, se podra recurrir, haciendo las modificaciones pertinentes, al sufrido INPUT, o incluso al comando INKEY$ o equivalentes.

Con estas líneas ya está incializado el programa, por lo que se pueden comenzar a codificar rutinas que cumplan las diferentes tareas a realizar. A continuación y tras una simple pantalla de presentación con el nombre del programa se puede ya empezar a construir el menú principal que dará entrada a las diferentes subrutinas que componen el programa.




*El Libro Mayor informatizado es más manejable y permite consultas más rápidas y eficaces que su equivalente impreso.*

## El menú de opciones

El cartel de entrada visualizará simplemente el nombre del programa, previo borrado de la pantalla. Tras un pequeño intervalo de retardo, ésta presentará directamente el menú principal. Naturalmente, el lector puede incluir aquí cualquier cosa que le dice su espíritu artístico; un simple cartel de entrada podría ser el siguiente:

```
60 CLS:POSITION 11,12:PRINT "GASTOS FAMILIARES"
70 FOR I=1 TO 500:NEXT I
```

El comando POSITION seguido del PRINT puede ser sustituido por un PRINT AT u otro comando equivalente. Para modificar el menú principal, será necesario determinar primero qué acciones se desean tomar a partir de él. Así, por ejemplo, éstas podrían ser las siguientes:

*Por medio de una opción incorporada al programa se puede conocer con celeridad y exactitud el estado actual de nuestra cuenta bancaria.*



1: Consulta del archivo, para ver asientos introducidos con anterioridad.
2: Actualización o entrada de nuevos datos.
3: Cargar del disco o casete un fichero de datos.
4.: Guardar el fichero en disco o casete.

Estas son las acciones básicas necesarias y suficientes para el objetivo que nos hemos marcado; si bien, el lector podría incluir aquí sus propias opciones con gran sencillez.

La rutina que imprime el menú en la pantalla es la que sigue:

```
90 CLS:POSITION 16,0:PRINT "OPCIONES"
110 POSITION 10,6:PRINT "1:ACTUALIZAR FICHERO"
120 POSITION 10,8:PRINT "2:CONSULTAR FICHERO"
130 POSITION 10,10:PRINT "3:CARGAR FICHERO"
140 POSITION 10,12:PRINT "4:GUARDAR FICHERO"
```

Su ejecución dará lugar a la siguiente pantalla:

```
        OPCIONES

1:ACTUALIZAR FICHERO
2:CONSULTAR FICHERO
3:CARGAR FICHERO
4:GUARDAR FICHERO
█
```

Ahora sólo queda por realizar la zona adecuada para captar la opción elegida por el usuario. Esto puede hacerse mediante el comando GET#1,M que espera a que se pulse una tecla y guarda su código en la variable M, a través del canal de entrada #1 previamente abierto para el teclado. De esta forma no es necesario pulsar la tecla RETURN como ocurriría con un INPUT. Después habrá que examinar si la opción elegida es válida y, si es así, enviar el flujo del programa hacia la subrutina correspondiente, lo que puede lograrse mediante el comando ON M GOSUB...

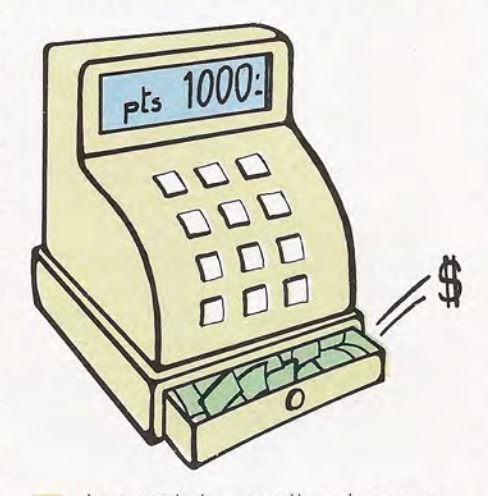
El listado de esta zona es la siguiente:

```
150 GET #1,M:M$=CHR$(M)
160 IF M$<"1" OR M$>"4" THEN 150
170 M=VAL(M$):ON M GOSUB 540,190,1380,1310
180 GOTO 90
```




*Si se añade al programa la posibilidad de imprimir los resultados puede confeccionarse un ejemplar de libro mayor que preserve nuestras cuentas de un posible fallo de la unidad de almacenamiento.*

Al recurrir a esta técnica es ineludible utilizar el comando CHR$, ya que el código almacenado en M debe transformarse en su carácter correspondiente para poder examinarlo. Así mismo, mediante VAL se transforma ese carácter en su equivalente numérico, necesario para el correcto funcionamiento de ON M GOSUB. En los números de línea a los que bifurca este último comando mencionado se situarán las correspondientes subrutinas, que no existen todavía, por lo que si se trata de ejecutar el programa se obtendrá un mensaje de error de línea no existente.




*La cantidad en metálico de que se dispone en casa se pude obtener como la diferencia entre el saldo total menos el de la cuenta bancaria.*

Como el orden de los factores no altera el producto, empezaremos ya a codificar la subrutina a la que se dirigirá el programa al pulsar el usuario la tecla "2"; ésta corresponde a la zona de consulta del fichero. Lo primero que hay que hacer es un borrado de la pantalla con CLS para eliminar el menú de opciones y, a continuación, formatear la pantalla con una cabecera que incluya los diferentes campos que componen cada asiento, para representar estos posteriormente. También hay que representar otras informaciones adicionales, como el número de asiento que señala el cursor, la página, el saldo...

```
190 CLS:PRINT "FECHA CONCEPTO CARGO IMPORTE"
200 POSITION 2,20:PRINT "PAGINA..."
210 POSITION 17,20:PRINT INT(PA/19)+1
220 POSITION 22,20:PRINT "ASIENTO..."
230 POSITION 34,20:PRINT PA+P-1
240 POSITION 2,21:PRINT "SALTO TOTAL...":POSITION
    17,21:PRINT SR
250 POSITION 2,22:PRINT "SALDO CUENTA...":POSITION
    17,22:PRINT SC
260 POSITION 2,23:PRINT "DIFERENCIA...";:POSITION
    17,23:PRINT SR-SC;
```

Con el comando POSITION (o con el PRINT AT en su defecto) se sitúan en su lugar correspondiente los diferentes mensajes; se obtendrá la siguiente pantalla:

```
FECHA          CONCEPTO          CARGO IMPORTE

PAGINA.....................1     ASIENTO .......... 1
SALDO TOTAL................0
SALDO CUENTA...............0
DIFERENCIA.................0
```

Al cambiar la página de la forma que se verá más adelante, cada mensaje borra el valor antiguo imprime el nuevo valor. Sólo queda ya imprimir los sectores del fichero que quepan en la página; debido a los mensajes anteriores, sólo podrán visualizarse simultáneamente 18 asientos:

```
270 POSITION 0,1
280 FOR I=PA TO PA+17
290 PRINT MID$(DATO$,I*40-39,I*40-1)
```

```
300 NEXT I
310 POSITION 7,P:PRINT ">"
```

La línea 310 imprime el puntero de modificación por confirmación del banco en el primer asiento de esa página; éste podrá ser desplazado por medio de las siguientes líneas del programa:

```
320 GET #1,M:M$=CHR$(M)
330 IF M<>45 AND M<>61 AND M<>43 AND M<>42
    AND M<>27 THEN 320
340 GOTO (M=45) *350+(M=61) *380+(M=42) *410+(M=27)
    *530+(M=43) *470
350 IF PA=1 THEN 320
360 PA=PA-18:P=1:GOTO 200
380 IF PA+18>A THEN 320
390 PA=PA+18:P=1:GOTO 200
410 POSITION 7,P:PRINT" ":P=P+1:IF P>18 THEN P=1
430 IF PA+P-1>=A THEN P=1
440 POSITION 7,P:PRINT ">":GOTO 200
470 VAR=(PA+P-1)*40-40:IF DATO$ (VAR+1)=" THEN
    320
480 POSITION O,P:PRINT " "
490 DATO$(VAR+1)=" "
500 IF DATO$(VAR+30)="D" THEN M$(1)="—":M$
    (2)=STR$(VAL(MID$(DATO$,VAR+32,VAR+39))):SC=
    SC+ VAL(M$ GOTO 200
510 IF DATO$(VAR+30)="H" THEN M$(1)="+":
    M$(2)=STR$(VAL(MID$(DATO$,VAR+32,VAR+39))):
    SC=SC+VAL(M$)
520 GOTO 200
530 RETURN
```

Aunque el funcionamiento de esta zona del programa puede parecer muy complicada a primera vista, no es así. Primero (líneas 320 a 340) se espera a que se pulse una tecla. El código obtenido variará mucho de unas máquinas a otras, por lo que es preciso consultar cada manual específico. Una posible correspondencia de códigos con las teclas pulsadas, puede ser la siguiente:

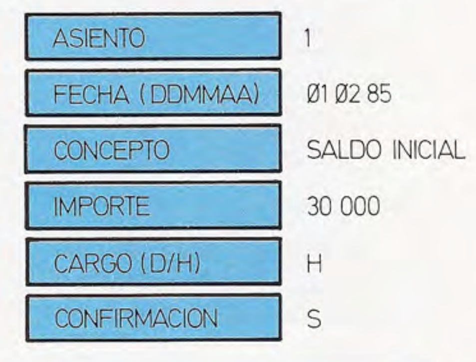
45: cursor hacia arriba
61: cursor hacia abajo
43: cursor a la izquierda
42: cursor a la derecha
27: tecla ESCAPE

Y esta es precisamente la utilizada en este programa. Así, al pulsar cada tecla se bifurcará con la sentencia de la línea 340 hacia la zona correspondiente, según la acción que se desee realizar. Estas acciones son las siguientes: con la tecla de movimiento del cursor hacia abajo se pasa a la página siguiente del fichero; con la del cursor hacia arriba, a la página anterior; con la del cursor a la derecha se desplaza hacia abajo el puntero de modificación (representado por el signo: ">"), de forma que al llegar el último asiento de esa página vuelve al primero de nuevo; y con la tecla del cursor hacia la izquierda se valida un asiento que no hubiera sido aún confirmado por el banco.

Al introducir los datos, los asientos no confirmados por el banco se identificarán por medio de un carácter asterisco (*) en su primer campo. De esta forma, en las líneas 470 a la 510, se busca si existe este asterisco en el asiento señalado; si es así se procede a su borrado y a calcular el nuevo saldo de la cuenta contabilizando ya el importe de ese asiento. Tras ello se vuelve a la línea 200 que imprime de nuevo estos valores actualizados. Pero, ¿cómo se sale de esta subrutina? Sencillamente, mediante la pulsación de la tecla de ESCAPE, volviendo de inmediato el control al menú de opciones.

## Entrada de datos

La siguiente subrutina que se va a codificar es la de entrada de datos (ver listado adjunto). Esta no ofrece ninguna particularidad especial, por lo que sólo se hará un breve comentario de su funcionamiento. La entrada de datos se hace a través del teclado y de acuerdo al procedimiento explicado anteriormente. Para ello se imprimen unos renglones que indican el concepto a intro-

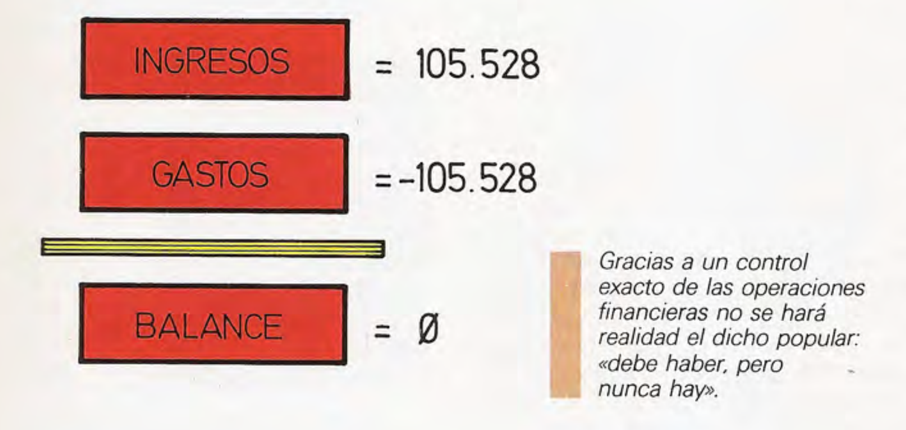



*La entrada de datos se realiza con el formato esbozado en este gráfico.*

| FECHA | CONCEPTO | CARGO | IMPORTE |
|---|---|---|---|
| Ø1Ø285 | > SALDO INICIAL | H | 30 000 |
| * Ø1Ø285 | INGRESO CUENTA | H | 16 000 |
| Ø1Ø285 | GASTOS VARIOS | D | 5 200 |
| Ø2Ø385 | COMIDA | D | 1 000 |
| | FIN DE DATOS | | |

PAGINA 1 — ASIENTO 1
SALDO TOTAL 39 800
SALDO CUENTA 23 800
DIFERENCIA 16 000

*Página de asientos que se obtiene al consultar el fichero de datos del programa.*




*Gracias a un control exacto de las operaciones financieras no se hará realidad el dicho popular: «debe haber, pero nunca hay».*




*Al recibir la confirmación de una operación por parte del banco, se puede confirmar el asiento, con lo que se actualiza el saldo de la cuenta bancaria.*

ducir en cada momento, lo cual se indica mediante el símbolo: ">". Así, para introducir la fecha, por ejemplo, se recurre a un bucle que se repite 6 veces, una para cada dígito. Los seis dígitos habrán de introducirse por tanto en la forma indicada: primero el día, seguido del mes y de las dos últimas cifras del año. Por ejemplo, el día 1 de abril de 1986, se introducirá con las siguientes cifras: 010486, no siendo necesario pulsar la tecla RETURN al final, pues automáticamente se pasa al campo siguiente. La longitud de los campos queda así limitada y al llegar a su final se pasa al siguiente campo.

Una vez introducidos todos los campos se pregunta si los datos son correctos, para en caso de detectar algún error poder corregirlo empezando de nuevo con la introducción de los datos. Estos datos se guardan temporalmente en las matrices respectivas, como se explicó al principio.

Si se dan por correctos, los datos serán transferidos al fichero formado por la matriz DATO$; esta función la realiza la siguiente zona del programa:

```
1180 VAR=A*40-40:A=A+1:FOR I=1 TO 40:DA
     TO$(VAR+I)=" ":NEXT I
1200 IF VALE$(1)="N" THEN DATO$(VAR+1)="*"
1210 FOR I=1 TO 6:DATO$(VAR+I+1)=STA$(FECHA(I)):NEXT
     I
1220 MID$(DATO$,VAR+9,VAR+28)=CON$
1230 LON=LEN(IMP$):MID$(DATO$,VAR+40-LON,VAR+39)
     =IMP$
1250 IF CARGO$(1)="D" THEN DATO$(VAR+30)=CARGO$
     :M$(1)="—"
1260 IF CARGO$(1)="H" THEN DATO$(VAR 30)=CAR-
     GO$:M$(1)="+"
1270 DATO$((A-1)*40+13)="FIN DE DATOS":DA
     TO$(501*40)=" "
1280 MID$(M$,2,2+LON)=IMP$:M=VAL (M$):SR=+M
1290 IF VALE$(1)="S" THEN SC=SC+M
1300 GOTO 540
```

Una vez introducidos los datos en la matriz DATO$, se empezarán a pedir datos para un nuevo asiento; éste es el momento, si se desea, de abandonar la subrutina para volver al menú de opciones. Al efecto se pulsará la tecla ESCAPE, cuya misión se detecta en la línea 690 en la que se ha indicado entre comillas el carácter "E"; este corresponderá con el carácter entregado por esa tecla en cada caso concreto, como ya se ha mencionado. Dicho carácter puede ser fácilmente sustituido para que abandone esta subrutina al pulsar cualquier otra tecla, como por ejemplo la "A", sin más que modificar la referida línea.

## Un archivo permanente

A estas alturas tan sólo restan por codificar las subrutinas de carga y grabación del fichero de asiento. Estas funciones se pueden realizar abriendo sendos canales de entrada o de salida, según se trate de almacenamiento o de carga, respectivamente.

Para ello se utilizará el comando OPEN (nos remitimos a lo expuesto anteriormente al abrir el canal de entrada para el teclado). Las subrutinas de carga y almacenamiento sólo se diferenciarán en que en una se utilizará el comando PRINT# para guardar el fichero en el disco a casete, y en la otra se utilizará el comando INPUT# para cargarlo en la memoria del ordenador. Ambos comandos deben trabajar con variables, por lo que se grabarán o se cargarán cada uno de los asientos por separado; ello se hará empleando la matriz M$ de utilidad general. Previamente, y para saber el número de asientos a cargar o grabar, se cargará o grabará la variable A que lleva la cuenta del número de asientos introducidos. Asimismo, se deben grabar los valores contenidos en SR y SC, correspondientes a los saldos que arrojan los asientos existentes.

Las subrutinas de carga y almacenamiento son las siguientes:

```
1310 CLS:PRINT "    CARGAR FICHERO"
```

```
540 CLS:PRINT "    ACTUALIZAR FICHERO"
558 IF A+1<501 THEN 590
560 POSITION 13,12:PRINT "FICHERO LLENO";:
    FOR I=1 TO 1000:NEXT I:RETURN
590 POSITION 1,5:PRINT " ASIENTO........":
    POSITION 18,5:PRINT A
600 POSITION 1,7:PRINT " FECHA
    (DOMMAA)."
610 POSITION 1,9:PRINT " CONCEPTO......"
620 POSITION 1,11:PRINT " IMPORTE......."
630 POSITION 1,13:PRINT " CARGO (D/H)...."
640 POSITION 1,15:PRINT " CONFIRMA-
    CION..."
650 POSITION 17,7:PRINT ">      ":POSI-
    TION 18,7
660 FOR I=1 TO
670 GET #1,M:M$=CHR$(M)
690 IF M$="E" AND I=1 THEN RETURN
700 IF M$="" THEN 670
710 IF M$<"0" OR M$>"9" THEN 670
720 PRINT M$;:FECHA(I)=VAL(M$):NEXT I
770 POSITION 17,7:PRINT " ":POSITION 17,9:
    PRINT ">":POSITION 18,9
790 CON$="":FOR I=1 TO 20
800 GET #1,M:M$=CHR"(M)
820 IF M$="" THEN 800
830 IF M=155 THEN 870
840 PRINT M$;:CON$(I)=M$:NEXT I
870 POSITION 17,9:PRINT " ":POSITION 17,
    11:PRINT ">":POSITION 18,11
880 IMP$="":FOR I=1 TO 8
900 GET #1,M:M$=CHR$(M)
920 IF M=155 THEN I=9:GOTO 960
930 IF M$<"0" OR M$>"9" THEN 900
940 PRINT M$;:IMP$(I)=M$
960 NEXT I
980 IF VAL(IMP$)=0 THEN 870
990 POSITION 17,11:PRINT " :POSITION 17,
    13: PRNT ">":POSITION 18,13
1000 GET #1,M:M$=CHR$(M)
1020 IF M$<>"D" AND M$<>"H" THEN 1000
1030 PRINT M$;:CARGO$=M$
1050 POSITION 17,13:PRINT " ":POSITION 17,
    15:PRINT ">":POSITION 18,15
1060 GET #1,M:M$=CHR$(M)
1080 IF M$<>"S" AND M$<>"N" THEN 1060
1080 PRINT M$;:VALE$=M$
1110 POSITION 17,15:PRINT " ":POSITION 8,
    20:PRINT "DATOS CORRECTOS (S/N) ?"
1120 GET #1,M:M$=CHR$(M)
1140 IF M$<>"S" AND M$<>"N" THEN 1120
1150 POSITION 0,20:PRINT "              "
1160 IF M$="S" THEN 1180
1170 GOTO 650
```

*PROGRAMA 1: Subrutina de entrada de datos.*

```
1320 GOSUB 1440:OPEN#2,8,0,M$
1330 PRINT #2;A:PRINT #2,SR:PRINT #2:SC
1340 FOR I=0 TO A:M$=MID$(DATO$,I*40+1,I*40+40)
1360 PRINT #2;M$:NEXT I: CLOSE #2:RETURN
1380 CLS:PRINT"    CARGAR FICHERO"
1390 GOSUB 1440:OPEN #3,4,0,M$
1400 INPUT #3;A:INPUT #3;SR:INPUT #3;SC
1410 FOR I=0 TO A:INPUT #3;M$:MID$(DATO$,I* 0+1
     I* +40)=M$
1430 NEXT I:CLOSE #3:RETURN
1440 POSITION 2,12:PRINT "DISPOSITIVO: FICHERO ";:IN
     PUT M$:RETURN
```

La subrutina de la línea 1440 es utilizada por las dos anteriores para obtener el nombre del fichero sobre el que se quieren grabar o cargar los datos; probablemente, el usuario tendrá que introducir modificaciones en ella para adaptarla al formato del comando OPEN de que disponga su intérprete BASIC.

```
10 DIM DATO$(501*40),M$(40),FECHA(6),
   CON$(20), IMP$'3),CARGO$'1),VALE$'1)
20 DATO$= ":DATO$(501*40)=" ":DATO$
   (2)=DATO$
30 LET A=1:LET PA=1:LET SR=0:LET
   SC=0:LET P=1
40 OPEN #1,4,0,"K"
60 CLS:POSITION 11,12:PRINT "GASTOS
   FAMILIARES"
70 FOR I=1 TO 500:NEXT I
90 CLS:POSITION 16,0:PRINT "OPCIONES"
110 POSITION 10,6:PRINT "1:ACTUALIZAR
    FICHERO"
120 POSITION 10,8:PRINT "2:CONSULTAR
    FICHERO"
130 POSITION 10,10:PRINT "3:CARGAR FI-
    CHERO"
140 POSITION 10,12:PRINT "4:GUARDAR FI-
    CHERO"
150 GET #1,M:M$=CHR$(M)
160 IF M$<"1" OR M$> 4" THEN 150
170 M=VAL(M$):ON M GOSUB 540,190,1380,
    1310
180 GOTO 90
190 CLS:PRINT " FECHA        CONCEPTO
    CARGO IMPORTE"
200 POSITION 2,20:PRINT "PAGINA........ "
210 POSITION 17,20:PRINT INT(PA/19)+1
220 POSITION 22,20:PRINT "ASIENTO.... "
230 POSITION 34,20:PRINT PA+P-1
240 POSITION 2,21:PRINT "SALDO TOTAL...
    ":POSITION 17,21:PRINT SR
250 POSITION 2,22:PRINT "SALDO CUENTA..
    ".POSITION 17,22:PRINT SC
260 POSITION 2,23:PRINT "DIFERENCIA....
    ;:POSITION 17,23:PRINT SR-SC;
270 POSITION 0,1
280 FOR I=PA TO PA+17
290 PRINT MID$(DATO$,I*40-39,I*40-1)
300 NEXT I
310 POSITION 7,P:PRINT ">"
320 GET #1,M:M$=CHR$(M)
330 IF M<>45 AND M<>61 AND M<>46
    AND M<>42 AND M<>27 THEN 320
340 GOTO (M=45)*350+(M=51)*380+(M=
    42)*410+(M=27)*530+(M=49)*470
350 IF PA=1 THEN 320
360 PA=PA-18:P=1:GOTO 200
380 IF PA+18>A THEN 320
390 PA=PA+18:P=1:GOTO 200
410 POSITION 7,P:PRINT " ":P=P+1:IF P>18
    THEN P=1
430 IF PA+P-1>=A THEN P=1
440 POSITION 7,P:PRINT ">":GOTO 200
470 VAR=(PA+P-1)*40-40:IF
    DATO$(VAR+1)=" "THEN 320
480 POSITION 0,P:PRINT"
490 DATO$(VAR+1)=" "
500 IF DATO$(VAR+30)="D" THEN
    M$(1)="-":M$(2)=STR$(VAL(MID$
    (DATO$,VAR+32,VAR+39))):SC=SC+VAL
    (M$):GOTO 200
510 IF DATO$(VAR+30)="H" THENM$(1)=
    "+":M$(2)=STR$(VAL(MID$(DATO$,VAR
    +32,VAR+33))):SC=SC+VAL(M$)
520 GOTO 200
530 RETURN
540 CLS: PRINT "    ACTUALIZAR FICHERO"
550 IF A+1<501 THEN 590
560 POSITION 13,12:PRINT "FICHERO LLE-
    NO";:FOR I=1 TO 1000:NEXT I:RETURN
590 POSITION 1,5:PRINT " ASIENTO........":
    POSITION 18,5: PRINT A
600 POSITION 1,7:PRINT " FECHA (DDMMAA."
610 POSITION 1,9:PRINT " CONCEPTO......."
620 POSITION 1,11:PRINT " IMPORTE......."
630 POSITION 1,13:PRINT " CARGO (D/H)...."
640 POSITION 1,15:PRINT " CONFIRMACION...
650 POSITION 17,7:PRINT ">
    ": POSITION 18,7
660 FOR I=1 TO 6
670 GET #1,M:M$=CHR$(M)
690 IF M$="E" AND I=1 THEN RETURN
700 IF M$="" THEN 670
710 IF M$<"0" OR M$>"9" THEN 670
720 PRINT M$;:FECHA(I)=VAL(M$):NEXT I
770 POSITION 17,7:PRINT " :POSITION 17,9:
    PRINT ">":POSITION 18,9
790 CON$="":FOR I=1 TO 20
800 GET #1,M:M$=CHR$(M)
820 IF M$="" THEN 800
830 IF M=155 THEN 870
840 PRINT M$;:CON$(I)=M$:NEXT I
870 POSITION 17,9:PRINT " ":POSITION
    17,-11:PRINT ">        ":POSITION 18,11
880 IMP$=" ":FOR I=1 TO 8
900 GET #1,M:M$=CHR$(M)
920 IF M=155 THEN I=9:GOTO 960
930 IF M$<"0" OR M$> "9" THEN 900
940 PRINT M$;:IMP$(I)=M$
960 NEXT I
980 IF VAL(IMP$)=0 THEN 870
990 POSITION 17,11:PRINT " ":POSITION
    17,18:PRINT "> ":POSITION 18,13
1000 GET #1,M:M$=CHR$(M)
1020 IF M$<>"D" AND M$<> "H" THE 1000
1030 PRINT M$;:CARGO$=M$
1050 POSITION 17,13:PRINT " ":POSITION
     17,15:PRINT "> :POSITION 13,15
1060 GET #1,M:M$=CHR$(M)
1080 IF M$<>"S" AND M$<> "N" THEN 1060
1090 PRINT M$;:VALE$=M$
1110 POSITION 17,15:PRINT " ":POSITION 8,20:
     PRINT "DATOS CORRECTOS (S/N)?"
1120 GET #1,M:M$=CHR$(M)
1140 IF M$<>"S" AND M$<>"N" THEN 1120
1150 POSITION 0,20:PRINT "
1160 IF M$="S" THEN 1180
1170 GOTO 650
1180 VAR=A*40-40:A=A+1:FOR I=1 TO 40:
     DATO$(VAR+I)=" :NEXT I
1200 IF VALE$(1)="N" THEN DATO$
     (VAR+1)="*"
1210 FOR I=1 TO 8: DATO$(VAR+I+1)=STR$
     (FECHA(I)):NEXT I
1220 MID$(DATO$,VAR+9,VAR+28)=CON$
1230 LON=LEN (IMP$):MID$(DATO$, VAR+
     40-LON,VAR+39)=IMP$
1250 IF CARGO$(1)="D" THEN DATO$(VAR+
     30)=CARGO$:M$(1)="-"
1260 IF CARGO$(1)="H" THEN DATO$(VAR+
     30)=CARGO$:M$(1)="+"
1270 DATO$((A-1)*40+13)="FIN DE DATOS":
     DATO$(501*40)=" "
1280 MID$(M$,2,2+LON)=IMP$:M=VAL(M$):
     SR=SR+M
1290 IF VALE$(1)="S" THEN SC=SC+M
1300 GOTO 540
1310 CLS:PRINT "        GUARDAR FICHERO"
1320 GOSUB 1440:OPEN #2,8,0,M$
1330 PRINT #2;A:PRINT #2;SR:PRINT #2;SC
1340 FOR I=0 TO A:M$=MID$(DATO$,I*40+1,
     I*40+40)
1360 PRINT #2;M$:NEXT I:CLOSE #2: RETURN
1880 CLS:PRINT "        CARGAR FICHERO"
1390 GOSUB 1440:OPEN #3,4,0,M$
1400 INPUT #3;A:INPUT #3;SR:INPUT #3;SC
1410 FOR I=0 TO A:INPUT #3;M$:MID$-
     (DATO$,I*40+1,I*40+40)=M$
1430 NEXT I:CLOSE #3:RETURN
1440 POSITION 2,12: PRINT "DISPOSITIVO:FI-
     CHERO ";:INPUT M$:RETURN
```

*PROGRAMA 2: Programa contable.*

# Indice Temático

## El ordenador útil



## Archivos en BASIC (I)



## Archivos en BASIC (II)



## Archivos aleatorios (I)



## Archivos aleatorios (II)



## Agencia telefónica



## Los archivos del Commodore 64

